Task: 4.2
Version: 1.4
Date: March 2004

Symbolic Solver Services. Wrapper Tools Release Candidate

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

ORCCA, University of Western Ontario

Deliverable D23 (Public)

Abstract

This deliverable re-package previous results from [9] and [10] and presents the Wrapper Tool that allows various advanced mathematical problem solving environments to be exposed through web services. In this document we describe all technologies designed and software implemented that were developed for Symbolic Solver and its wrapper tool. Those include an approach to portable mathematical server architecture, software for maintaining the life cycles of mathematical web services, and client-side facilities to access and use services deployed. We also detail mathematical services implemented within the task 4.2.

Contents

1 Architecture of the Wrapper Tool for Symbolic Services

The idea of the Symbolic Solver Wrapper Tool architecture is to provide an environment that will encapsulate advanced mathematical software packages, such as Maple[23], Axiom[24], Derive[25], Mathematica[26] or Matlab[27] and expose the functionalities of these systems through symbolic services deployed.

1.1 Symbolic Solver Environment

The main concept of the Symbolic Solver assumes that all developed MONET symbolic services are running as independent web services, each reachable at its own unique URL and offering its own functionalities. All of these services are enclosed within a specially designed software engine, called *Monet Symbolic Server* and they are managed by a wrapper tool named *Symbolic Solver Environment*.

The general scheme of organization for the Symbolic Solver Environment is shown in Figure 1. It demonstrates that each Monet symbolic service is assigned to several instances, such as service core Java class, source code implementing the service with a mathematical solving software[1] (usually a computer algebra system), and MSDL [7]. The principal information about each service is provided by the service configuration file that contains tree parts: service MSDL, service interface to mathematical solving system(s) and the actual service implementation with those systems. The wrapper tool is a container for all symbolic services, which is masked by their web interfaces and carries out their main functionalities, including installation and invocation.

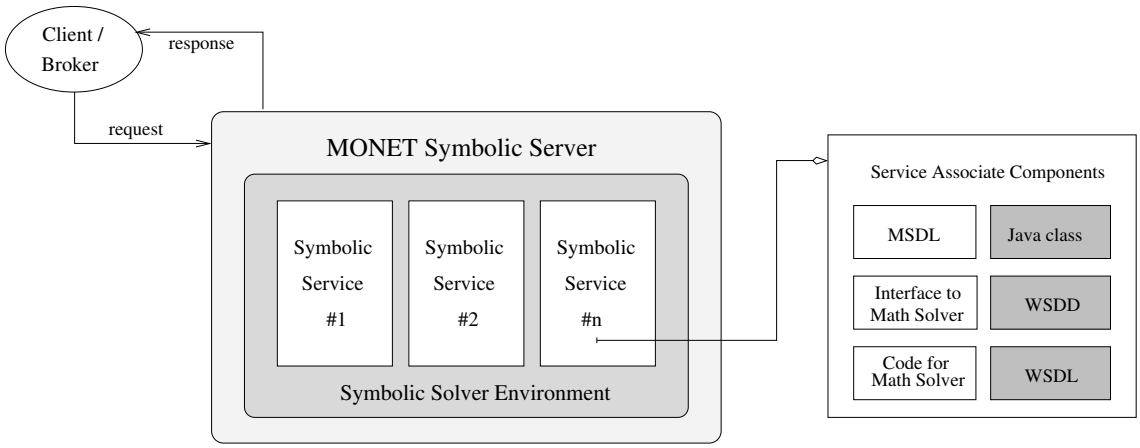

Figure 1: The general scheme of Monet Symbolic Server Architecture

There are two core software tools to maintain this wrapper environment: one is responsible

[1] Hereafter called *mathematical solver system* or *mathematical solving engine*

for service creation and installation, and the other for service invocation. Both of them will be discussed later in the section "Implementation".

1.2 Symbolic Service Organization

The Symbolic Services are organized in the way that the author of a new service does not need to know about web servers and services, Java or various XML technologies in order to create a new service. Instead the service's author is asked to provide the code that implements the service using the language of the mathematical system chosen to serve as solving engine of the service. The author aslo has to indicate the name of the main function that calls that implementation part.

As shown in the Figure 1, each service is associated with three original pieces of information: MSDL, service interface to the mathematical solving system, code written using language of that software and with three generated: Java class, web service deployment descriptor (WSDD) [13] and web service description language file (WSDL) [12].

1.2.1 Service Configuration File

All original information about symbolic service is provided by a service author and stored in an XML-based configuration file. This file is the item in the Symbolic Solver Architecture that uniquely and completely represents the service itself. This file consists of three parts: service MSDL skeleton(to be complete automatically during service installation), service interface to mathematical solving engine (computer algebra system for example) and service implementation (code written in the language of the solving system). One file can contain descriptions for more than one service, especially it makes sense when several services share parts of implementation. The structure of the configuration file has the following pattern:

```
<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"/>
        {interfaces for other services}
    </services>

     <implementation language = "math_solver_name">
        {implementation for each service using
```

```
        the language of the corresponding solver}
    </implementation>
</mathSever>
```

An example of configuration file for limit calculation service is given in the section 10.2.2 The configuration file for each service should be available for the wrapper tool at any time after the service is installed on the Monet Symbolic Server.

1.2.2 Generated Service Associates

If 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, WSDD and WSDL files. All of them are automatically generated by the Symbolic Solver Environment installation engine, according to the information provided in the service configuration file.

Those generic components are basically used to deploy, expose and run service on the Monet Symbolic Server. 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. Generated WSDD is used to deploy service on the Monet Symbolic Server, and WSDL is exposed to the outside world to provide Monet brokers and clients with the information about service interface (see Figure 2).

1.2.3 Realization of Service Functions

A symbolic service mathematical functionalities is carried out by a combination of service Java class, Symbolic Solver invocation tool and information from service configuration file.

The main idea of the service infrastructure is that its Java class receives all SOAP requests from the outside (client or brokers), extracts information about mathematical arguments passed with the request (if any) and then calls Symbolic Solver Environment invocation engine. Symbolic Solver Environment in its turn creates program for the solving engine in real time, using information from service configuration file and mathematical arguments from a service call. This program is passed to solving software for execution, the result is retrieved by the same Symbolic Service Environment tool, wrapped into a SOAP message and sent back to the client or broker (see Figure 3).

1.3 Advantages of the Architecture

1.3.1 Portability

- **Service.** The described approach to service organization allows services to be portable:

once a configuration file for the service has been created, it can be used for this service installation on another Monet Symbolic Server.

- **Server.** The software package, designed to maintain the Monet Symbolic Server also provides it with a complete portability. As it will be shown later, the Monet Symbolic Server can be very easy installed on a web server, running Tomcat[17] and Axis[19].

1.3.2 Flexibility

- **Service.** Introduced way of service organization also ensures service flexibility: any service functionalities can be changed or updated by adjusting of the implementation part (program code for mathematical solver) in the service configuration file.

- **Server.** Installation tools for Monet Symbolic Server allow to choose its configuration, depending on the system environment and user preferences. More details about server installation will be described in the section "Implementation".

1.3.3 Extensibility

- **Service.** Service functionalities can be extended because of natural extensibility of the technologies, used in its implementation. If one needs, new CD can be added to OpenMath; new modules or packages can be written to expand build-in routines of computer algebra system. Indeed both of those opportunities were used in the current Symbolic Solver implementation: in Series Expansion Service new OpenMath CD for power series was added; new Maple package was created for Fractional Order Differentiation Service.

- **Server.** The architecture of Symbolic Solver provides the Monet Symbolic Server with a desirable extensibility: new services can be easily added to the Symbolic Solver Environment by using its installation engine.

To demonstrate the idea of Symbolic Solver Server architecture we offer two slightly different implementations by the University of Bath and the University of Western Ontario. Their common functions and specifications, as well as differences in the realization were discussed in [10] and also will be mentioned in the course of this document.

2 Technologies Used

Most of technologies used in the MONET project were presented in [1]. We included reasoning explaining why those approaches were used in [4]. Here we will briefly re-name those of technologies used for Symbolic Solver Services implementation.

2.1 Linux

It was decided to use Linux operation system to run Monet Symbolic Server, since Linux is known to be reasonable stable; it allows dynamic and remote control, its shell scripting is sophisticated enough to maintain large project. Furthermore this OS is relatively secure and mostly virus-protected. However client of Symbolic Services can be ran under other operation systems such as SUN Solaris or Windows.

2.2 Java

The main software for Symbolic Solver Environment is written in Java [16], using various libraries and applications also having Java API (RIACCA OpenMath library, Axis, Tomcat — see below)

2.3 Axis

Apache Axis [19] is a web servlet used to wrap all of developed Symbolic Services. It allows easily deploy web service, by using web service deployment descriptor(WSDD), that we provide for each of our symbolic services. Axis has its own web interface to browse list of deployed web services and associated with them WSDLs [12] We also use Axis's features of TCP and SOAP monitoring to capture messages flowing between Monet Symbolic Services and their clients. In our case Axis is running within Tomcat servlet container.

2.4 Tomcat

Apache Tomcat [17] is used as a web application server to provide a container for Monet Symbolic Server. All developed services (as part of Axis servlet) and their web clients (as independent Java Servlet Pages [20] are running under Tomcat.

2.5 SOAP

Simple Object Access Protocol is mainly used by Axis to exchange messages between various parts of MONET architecture: Monet Services(both symbolic and numeric), Monet broker and a whole variety of Monet clients.

2.6 WSDL

WSDL is used to provide Monet services clients with information about service interface. We use combination of both manually created and automatically generated (by Axis) WSDLs for developed services. WSDL plays an important rôle in service binding part of MSDL, which explains how to map parts of Monet queries(MSQL)[6] into proper parts of service input data.

2.7 JSP

Technology of JavaServer Pages applied to design client-end web interface, that easies service invocation, and in case of Bath implementation provides tools for symbolic service installation.

2.8 MSDL

The MONET approach to describe mathematical services [7] is used for it direct purpose. Each of developed services is associated with it own MSDL file, used then to advertise this service with the MONET Broker

2.9 Computer Algebra Systems

In Wrapper Tool different computer algebra systems may serve as solving engine, which task is to provide mathematical functionalities of symbolic services. Maple[23] was chosen as an example of the solving engine for the first implementation of Wrapper Tools. Being a advanced extensible computer algebra system and programming language at the same time Maple allows a wide variety of mathematical problem to be solved by using its build-in routines, as well as additionally created packages. Another computer algebra system Axiom[24] was then used to validate the Wrapper Tool architecture and to demonstrate abilities of the Symbolic Solver Environment to adopt different solving engine without performing major changes to its structure.

3 Implementation

3.1 Tools and Requirements

Symbolic Solver Environment implementation involves

- Java programs and libraries
- shell scripts
- JavaServer Pages
- XSLT stylesheets
- Computer algebra system(s) to be used as solving engine(s).

The main requirements for the system to run Monet Symbolic Server and Symbolic Solver Environment are

- Linux Redhat, version 7.0 or later
- Java SDK, version 1.4 or later
- Apache Tomcat
- Apache Axis
- Apache Ant
- Installation of a computer algebra system chosen(in case Maple version 8 or later).

All of the above software should be properly installed and running on the system where Symbolic Solver Environment is going to be set.

As it was mentioned in 1.1 the implementation of Symbolic Solver Environment includes two main parts: installation and invocation managers. The first of them takes care about new service creation and maintaining (installation/deinstallation). The second is a engine that handles calls to service from the outside. There is also a third component of Symbolic Solver Environment, which is responsible for installation of Monet Symbolic Server.

3.2 Monet Symbolic Server Installation

Monet Symbolic Server installation process involves setting up an existing web server in order to prepare it for running Monet symbolic services. We assume that Tomcat and Axis are running on a local host on a port, which can be accessed from the outside of the local network.

Once all necessary software listed in 3.1 are properly installed, the administrator of the Monet Symbolic Server implemented by UWO can execute the shell script `mathserver_init.sh` that performs instant Monet Symbolic Server installation. This script is due to perform the following steps: check the system settings and availability of all required components, then

run Apache Ant task to build Symbolic Solver java libraries from the source code included in the deliverable package and place them in proper locations in the Apache Tomcat and Apache Axis directory trees. Auxiliary java libraries such as RIACA for encoding OpenMath, RISC for serialization of RIACA objects and codecs for OpenMath to solving system phrasebook will be also copied to Axis library directory. At the end Axis servlet will be automatically restarted.

The architecture of Monet Symbolic Server is designed to be flexible, which allows to set it on Tomcat web servers with various configurations. Together with installation script provided it makes the Symbolic Server truly portable.

Bath implementation of the Monet Symbolic Server does not offer an installation script, but rather provides detailed instructions how to set web server to run Symbolic Services manually, which also allows this version of Symbolic Service to be portable.

3.3 Symbolic Service Installation

The UWO Service installation manager is implemented as a combination of Java programs and shell scripts. Each of Java programs is responsible to carry out one or more steps of service installation process, which in whole is driven by manager shell script. As described in 1.2 all necessary information about service is provided in its configuration file by the author. Therefore in order to install service the installation manager needs access to this file. To register a new service with the Monet Broker, the installation manager also requires the URL of the Broker. Optionally service administrator may specify port on which service will be running, if different from default and a path to the installation of the main solving software, if it has not been specified in the system path.

The general interface of the service installation manager call looks like

```
mathservice_init.sh <service config.file> <broker URL> [port] [path to solving software]
```

While installing a new service(see Figure 2), 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 WSDD,
- deploys the service on the web server,
- retrieves WSDL file generated by Axis after the service deployment,
- updates MSDL according to generated WSDL: it adjusts service binding part and fills-in the interface to client and broker elements.

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.

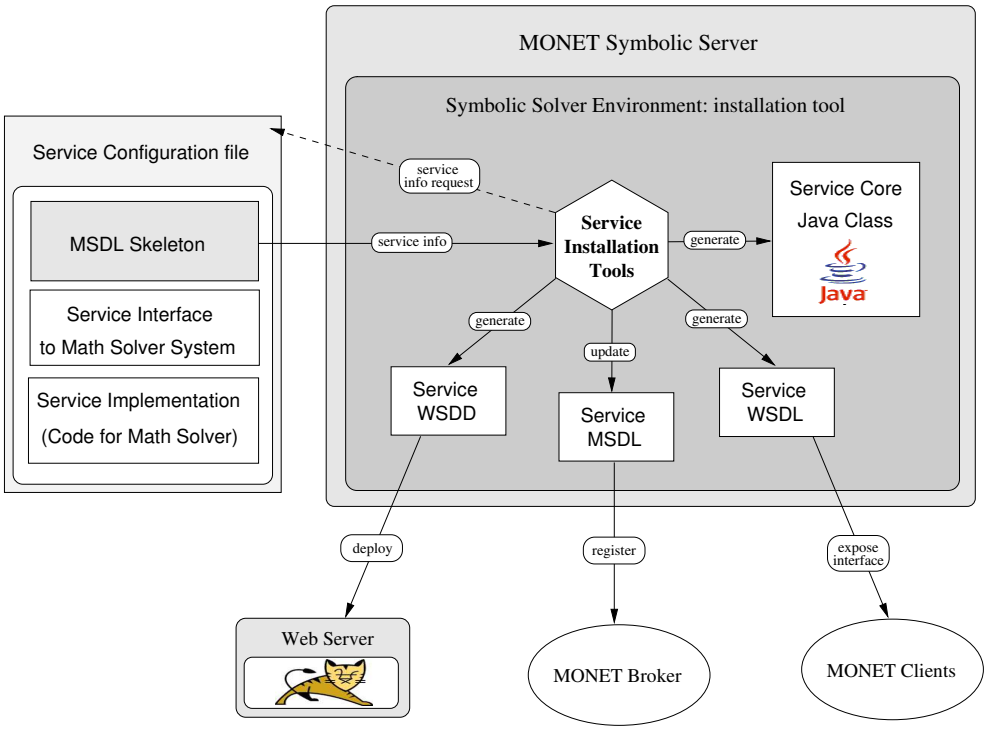

Figure 2: Monet Symbolic Service Installation

In case of Bath implementation of Symbolic Solver Environment, service installation manager is driven by JavaServer Pages, running as a separate web service, rather then by shell scripts. This allows remote service installation over the Internet, which beside obvious advantages hides the risk of possible damage of the web server by incompetent or invalid services installations. Furthermore all data for service configuration file should be re-entered in forms provided on the JSP pages every time service is deployed. However Bath implementation includes the feature of on-line MSDL generation, which is really helpful, especially for services handling large mathematical problems.

3.4 Symbolic Service Invocation

When symbolic service receives a SOAP request with a mathematical query, the service core java class retrieves the input arguments from this query and calls the service invocation manager.

The Symbolic Solver Environment service invocation manager is implemented as a combination of several Java libraries and auxiliary packages to computer algebra systems, designed to fulfill symbolic service functionalities, according to the information provided in the service configuration files. The following algorithm (also shown as scheme in the Figure 3)is

executed every time a symbolic service gets invoked:

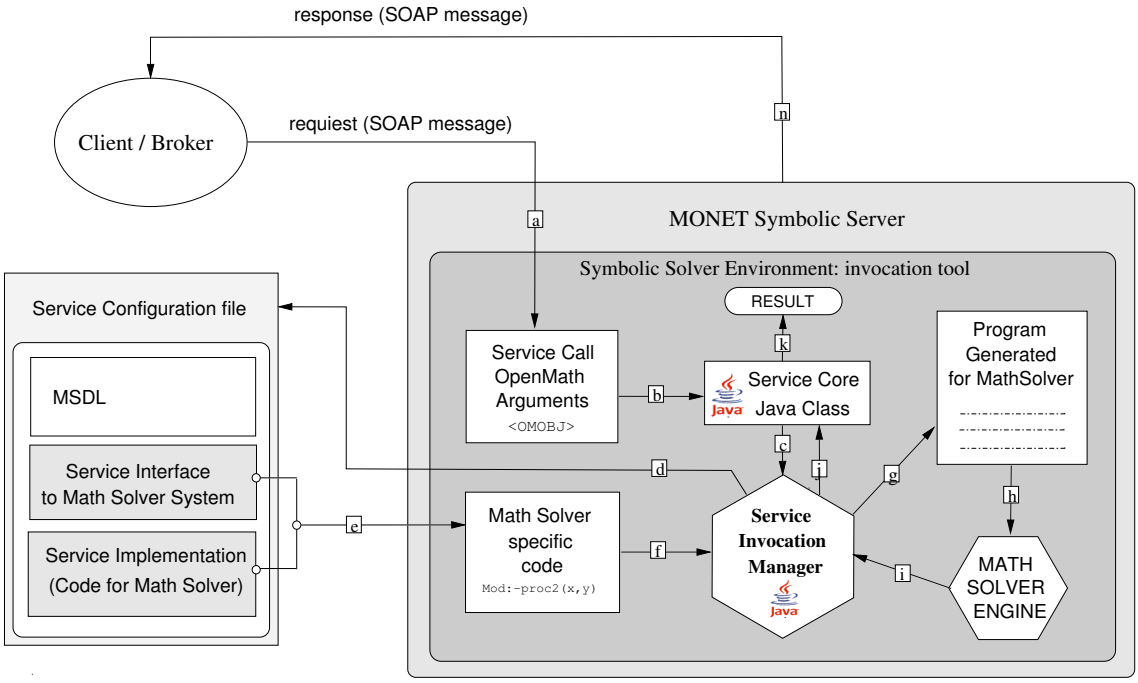

Figure 3: Monet Symbolic Service Invocation

1. Service Java class calls the invocation manager with the following parameters
   - the reference to the service configuration file
   - an array of mathematical arguments from query to the service
   - (optional) mathematical formats, if client set preference for encoding of service inputs and outputs (see section 5).
   - location of mathematical format conversion tools, defined for this particular service during its installation (for example, the service administrator may choose to use special XSLT stylesheets for Content MathML to OpenMath transformation or particular software package for translation between OpenMath and syntax of a computer algebra system used as a solving engine).
2. Invocation Manager parses configuration file and extracts service implementation and service interface to the solving mathematical software.
3. Service arguments gets parsed to strings representing OpenMath expressions or converted to syntax of mathematical solving system of by using a phrasebook specially designed for this particular system (depends on the implementation of Symbolic Solver and also on preferences set by client).

4. Invocation manager **generates** program to be run by the mathematical solving engine, based on the implementation part from the service configuration file, service interface to this system and mathematical arguments from service request.
5. The generated code is passed to the solving engine (for example a computer algebra system) for execution.
6. The results from solving engine returns to the invocation manager, there it gets converted into appropriate mathematical format and encoding(see section 5.2, afterwards it is passed back to service core java class.
7. Service Java class wrappers the answer into SOAP message in sends back to service client.

4 Replacing Mathematical Solving Engine

As has been mentioned in the section 1.2.1 different problem solving environments can be used as mathematical solving engine of the Wrapper Tool.

The approach to the organization of the Symbolic Solver Environment allows to reuse its architecture for developing symbolic services using different mathematical software as solving engine.

From the point of view of the service author switching between mathematical solvers means *only updating the service configuration file*. From the point of view of the Symbolic Solver Environment developer it means replacing components of Invocation Manager that are specific to the mathematical solving software used, however the second part of Symbolic Solver Environment – Installation Manager remains the same.

4.1 Changes to service configuration file

The idea of configuration file allows to specify the solving system name and the service implementation with this system. It also permits to have more than one implementation for the same service, using alternative solving engines. For example the service for definite integration may use system Derive, Axiom or Mathematica instead of Maple or 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 Maple, Axiom and Derive software for definite integration service, configuration file could 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>

</mathServer>
```

4.2 Changes to implementation Symbolic Solver Environment

In order to enable capabilities of Symbolic Solver Environment to handle services that use other computer algebra software, the Symbolic Solver Environment developer has to replace components of Invocation Manager, which depend on the computer algebra system, such as code generation means, tools for conversion between OpenMath and language of this system and an adaptor that allows to plug the system software into Symbolic Solver Environment.

Essentially it means providing three new java classes, by default named as

- `<solver_system_name>ServiceImpl`
- `<solver_system_name>CodeGenereation`
- `<solver_system_name>Output`

The stub code of those classes for known solving systems(such as Maple, Axiom, Mathematica, etc.) may be already written and stored in known locations. In this case the developer only has to point at those locations while installing the Symbolic Solving Environment. Otherwise those classes should be written from scratch.

However the whole scheme of Symbolic Solver Environment as well as tools of Symbolic Solver Environment assigned to install and maintain new services remain the same, since they do not depend on the solving engine.

5 Service input and output

5.1 Input Format

All symbolic services accept input arguments in OpenMath format. Each service takes an array of OpenMath expressions, that can be encoded as

- plain string
- XML DOM objects[21]
- RIACA OpenMath object[15]

The reason to support all of those encoding was to provide a choice of possible interfaces to clients and brokers by the same service.

String is a simplest way to format a request, which 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.

XML DOM is a moderate stage of encoding OpenMath objects that at least can ensure that the passed arguments represent a valid XML.

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

5.2 Output Format

In general case services return string object, encoding OpenMath expression, which contains the list of results or error message (see 6.2).

5.2.1 Representing multiple outputs

By default in case of success Symbolic services return answer, wrapped in an OpenMath list object.

This decision was made to unify the output from those symbolic services, which can return more than one result. For example the output from Root Finding Service with the input $e^x - a = 0,\ a > 0$ is $\ln a$ but for the input $(x^2 - a) = 0,\ a > 0$ the answer has 2 entries $\sqrt{a}$ and $-\sqrt{a}$. In both cases the output will be put in a list:

- $[\ln a]$

```
<OMOBJ xmlns = 'http://www.openmath.org/OpenMath'>
 <OMA>
   <OMS cd = 'list1' name = 'list'/>
     <OMA>
     <OMS cd="transc1" name="ln"/>
     <OMV name="a"/>
   </OMA>
 </OMA>
</OMOBJ>
```

- $[-\sqrt{a}, \sqrt{a}]$

```
<OMOBJ xmlns = 'http://www.openmath.org/OpenMath'>
 <OMA>
   <OMS cd = 'list1' name = 'list'/>
   <OMA>
     <OMS cd="arith1" name="root"/>
     <OMV name="a"/>
   </OMA>
   <OMA>
     <OMS cd="arith1" name="unary_minus"/>
     <OMA>
       <OMS cd="arith1" name="root"/>
       <OMV name="a"/>
     </OMA>
   </OMA>
 </OMA>
</OMOBJ>
```

5.2.2 MEL format

Upon a client request, the answer from the service can be wrapped into a service response object, according to the Monet MEL ontology (Mathematical Explanation Language) [3]. In this case the output looks like

```
<executionResponse xmlns="http://monet.nag.co.uk/monet/ns"
   href="http://ptibonum.scl.csd.uwo.ca:16661/axis/services/LimitCalculationService">
   <explanation>
       <errorcode>1</errorcode>
       <explanationFormat isProvided="true"
           href="http://monet.nag.co.uk/monet/explanation#expl1">
           <myexplanation xmlns="http://www.orcca.on.ca/MONET/explain">
               <Algorithm>
                MONET_limit_module := module()
                  export monet_limit;
                  monet_limit:=proc(function,var,limit_point);
```

```
                limit(function,var=limit_point);
              end:
             end:
            </Algorithmn>
            <ExecutionTime>
                12s
            </ExecutionTime>
        </myexplanation>
    </explanationFormat>
</explanation>

<result>
    <resultFormat href="http://monet.nag.co.uk/monet/result#openmath">
        <OMOBJ xmlns = 'http://www.openmath.org/OpenMath'>
           <OMA>
             <OMS cd = 'list1' name = 'list'/>
             <OMI>1</OMI>
           </OMA>
        </OMOBJ>
    </resultFormat>
</result>

<resultAdditionalInfo>
    <addInfoFormat isProvided="true">
        <additionalInfo xmlns="http://www.orcca.on.ca/MONET/addInfo">
            <accuracy>
                10E-7
            </accuracy>
            <logfile>
               http://ptibonum.scl.csd.uwo.ca:16661/logs/mathservice48486.log
            </logfile>
            <input_encoding>
              RIACA OMOBJ
            </input_encoding>
            <output_encoding>
              String
            </output_encoding>
        </additionalInfo>
    </addInfoFormat>
 </resultAdditionalInfo>
</executionResponse>
```

We do not offer this option by default, since when the result from a service is sent back to the Monet Broker, the last will provide the wrapping, possibly including supplementary information, such as explanation from a Planning Manager or Execution, etc.

5.2.3 Additional mathematical formats for service output

In addition, UWO Symbolic Service implementation supports several other mathematical formats to encode service results. User may set his or her preference while calling a service, and the answer will come in one of the following formats:

- OpenMath
- Content MathML
- PresentationMathML
- LaTex

This is an experimental facility that can be used in various environments, which integrate several formats to represent mathematical objects.

5.3 Service API

Service programming interface is presented by service WSDL file. This file should be accessible by clients and brokers, so they can create a proper service call. By default the WSDL file for each service is located at `<configFile>?wsdl` .

In order to support all of encoding for OpenMath objects, listed in 5.1, each service has several API plug-ins. Basically main operation offered by services are provided by run and check methods, that accepts service arguments in three different formats:

```
String checkService (String[] mathArgs )
String runservice(String[] mathArgs)

String checkService (org.w3c.dom.Node[] mathArgs )
String runservice(org.w3c.dom.Node[] mathArgs)

String checkService (nl.tue.win.riaca.openmath.lang.OMObject[] mathArgs )
String runservice(nl.tue.win.riaca.openmath.lang.OMObject[] mathArgs)
```

6 Service error handling

It is always important not only to create a robust software tool, implementing one or another sophisticated technology, but also to foreseen a range of possible problems and exceptional situation, when this tool may have problems or even fail to carry out its functions. It means we have to provide the developed product with a reliable error handling mechanism, that will protect the software environment from crashing or becoming irresponsible in unusual circumstances.

This error-handling tool should perform two main functions: firstly, ensure software stability in case of unpredicted conditions and secondly, provide legible explanation about possible cause of the arisen exceptional situation.

6.1 Error catching during server and service installation

Service installation toolkit of the Symbolic Solver Environment offers a simple, but still helpful error catching utility. Both server and service installation scripts, while running, check exit codes of all shell commands, performed on each step of the installation process. They also monitor system error and output streams. In case an exception arises, the installation process will be stopped and an appropriate to the situation message will be printed on a screen.

Installation manager automatically redirects error stream to a file `error.log` in its home directory, so the administrator of the Symbolic Server can browse it later to spot the reason of the trouble. The Figure 4 shows one of possible error situations during a new service installation.

6.2 Errors catching during service invocation

We distinguish two main types of errors and exceptional situation, that can happen during service calls: software/hardware problems and actual symbolic solver-related errors.

Among software/hardware we can list for example the following

- **connection fails**
  Those include "host/target/URL not found", "connection refused", "connection time-out", "wrong protocol", etc.
- **input/output exceptions**
  Typically they are "file not found", "read/write permission denied", etc.

Since Java is the main engine running symbolic services, those types of errors are relatively easy to catch, because of well developed exception handling mechanism in Java. If any of listed above or similar errors occur, the corresponding exception will be caught and proceed

```
<..snip..>
Deploying service SystemEquationSolveService on the web server:
Web server settings:
   host name = ptibonum.scl.csd.uwo.ca
   port number = 8081
<..snip..>
Tomcat manager username and password are set properly
Reloading Axis ...
<..snip..>
RELOAD FAILED
Axis Connect Exception: Connection refused
Total time: 2 seconds
--- *ERROR* -----------------------------------
Some errors occurred while reloading axis.
Please check the host name and the port number.
For detailed information see error log in
/usr/MonetSymbolicServer/error.log
-----------------------------------------------
```

Figure 4: An error message from Symbolic Solver Environment installation manager. The user tried to install a new symbolic service on a wrong port of the web server.

by Java Core of service invocation tool. The content of the exception then will be wrapped into a SOAP message and sent back to a client.

It is important for error reporting to distinguish the hardware/software caused errors from ones that occur while service is trying to solve a mathematical-related problem.

In other words, we have to catch errors, raised within mathematical solver and report them separately.

The following parties can be potential sources of problems related to mathematical solving system:

- The author of the service, who provides code written for mathematical solver to implement the service. Usually authors validate their programs, before submitting it for MONET services; therefore solver-related errors in service implementation parts are rare, but still not hundred percent excluded. We found that the most frequent issue of the problem in author distribution to the symbolic service is mismatching service interface and service implementation parts in the configuration file. This error is easy to correct but sometimes hard to notice, since it will not be flagged by solving system during the test run of the code written for service implementation. For example if someone designs services for simple arithmetical functions, describing their interfaces to maple as

  ```
  <services>
   <service name="MultService"  call="My_arithm1_module:-my_mult"/>
  ```

```
 <service name="DivService"   call="My_arithm1_module:-my_div"/>
</services>
```

and putting in the implementation part

```
<implementation language = "maple">
   My_arithm_module1 := module()
     export my_div, my_mult;
    <..snip..>
</implementation>
```

then the Monet Symbolic Solver will fail to fulfill any request for both of this services, because the Maple program, generated by Solver Environment will contain an expression `My_arithm1_module:-my_mult(param1,param2)` , which will not evaluate, since module `My_arithm1_module` has not been defined in the implementation part.

- Client. Potentially clients may submit queries for a service with wrong number or type of arguments. If those errors are not caught by Monet Broker, the service will receive a request, which will be translated to an expression that is invalid in syntax of mathematical solving system. For example one by mistake can call a definite integration service with missing upper bound of integration or a partial differentiation service with a number instead of differential variable.
- Service implementation routines: OpenMath to mathematical solver language phrasebook and solver-specific program generation tools. Theoretically extensions of OpenMath to mathematical solving system conversion tools, written by Symbolic Solver users may contain typos or wrong solver-specific entries. For example if mapping for OpenMath object

```
<OMOBJ>
  <OMA>
    <OMS cd="calculus1" name="int"/>
    <OMS cd="transc1" name="sin"/>
  </OMA>
</OMOBJ>
```

is set to be Mathematica expression `Integrate[sin]` , Mathematica processor will raise an exception while parsing this statement, because the proper syntax for indefinite integration operator in Mathematica is `Integrate[&function,&variable]`

Catching Errors within Mathematical Solving Systems

Most errors occurred during execution of the service code with the mathematical solving systems should be caught and handled within these systems.

The error catching mechanism may be different for each mathematical solving system. Therefore when switching from one mathematical solving engine to another, the developer of the Symbolic Solver Environment has to provide a code fragment with error catching methods, specific to of the mathematical solving system used. This code is meant to be included into the stub of solver-specific program that will be used each time services of this Symbolic Solver Environment are called.

In [10] we suggested the way to catch error in Maple by using build-in `try-catch-finally` statement. We assume the similar capacities of other computer algebra systems be available to develop Symbolic Solver Environment on their platforms.

The main task of the error handling tools is to provide a legible explanation about the error nature. By default the error message from the mathematical solver should to be converted into OME (OpenMath Error) object. The result OME then will be wrapped into SOAP message and sent back to client or broker, so the caller of the service will get a valid OpenMath expression, explaining the reason of the service fail.

For example if client calls a differentiation service with parameters $\tan(x)$ and 1, the answer from the service might be

```
<OMOBJ xmlns = 'http://www.openmath.org/OpenMath'>
  <OME>
    <OMS cd = 'moreerrors' name = 'algorithm'/>
    <OMSTR> wrong number (or type) of arguments </OMSTR>
  </OME>
</OMOBJ>
```

Another very common error case is runaway argument: for example if user submits a request for limit calculation service without specifying a limit point, the Symbolic Solver has complain about missing parameter:

```
 <OMOBJ xmlns = 'http://www.openmath.org/OpenMath'>
  <OME>
    <OMS cd = 'moreerrors' name = 'algorithm'/>
    <OMSTR> MONET_limit uses a 3rd argument, limit_point, which is missing </OMSTR>
  </OME>
</OMOBJ>
```

For now we are able to use only one OpenMath CD to encode service execution errors. In a future we hope to have more OpenMath errors CDs for, in order to distinguish syntax-related error (such as in a previous examples) from computation errors (such as division by zero).

7 Providing Explanation From Service

It has been a try to provide a user with the additional information from the service, such as used methods, performed computation and trace of called functions.

The pervious deliverable [10] contains a description of providing the explanations from a service, by using overloaded Maple method that returns the user information. The similar approach can be applied to other computer algebra systems: its main idea is in retrieving and re-defining, if necessary, the information provided by the kernel of solving system to user.

Usually the author of the service is the one who provides service with meaningful user information. It is more reasonable to include such explanations into service implementation part of a configuration file, than in a generated solver-specific code.

The main use of this user information is to provide an explanation from a service that can be sent to a client within service response message, constructed according to MEL [3].

8 Service Monitoring

8.1 Service Log Information

Symbolic Solver Environment creates log files for each service invocation. The log file contains records about each step of service execution, starting with loading service from configuration file, parsing arguments from a request, creating code to be run with mathematical engine, passing it for execution to the mathematical solving software and retrieving the result of computation. Each entry of log file is preceded by the time stamp. Keeping such a log allows to browse history of service invocation step by step, it gives an idea about time spent for each step of execution, and in case of exceptional situation or problem with fulfilling service request, this log information can help to spot the cause of trouble. The example of invocation log for Limit Calculation service can be found in Appendix 12

8.2 SOAP monitoring

In addition to logging of service execution, we also allow to monitor SOAP messages floating between service and its client. Both request to service and response from it is caught by Apache Axis SOAP Monitor. In order to enable the monitoring, all services have to be specially configured, when deployed with Apache Axis web servlet. The tools and utilities to enable symbolic services handling by the SOAP monitor is provided by the Symbolic Solver Environment distribution. They can be found in directory `axis_tools` .

The SOAP monitor utility itself is accessible with a web browser by going to `http://<host>:<port>/axis/SOAPMonitor` , where `<host>` and `<port>` are correspondingly the host name and port number where the Monet Symbolic Server is running.

In case administrator of the symbolic server does not permit access to SOAP monitor from clients, they can monitor SOAP messages flowing between them and services by using service-client interface toolkit, supplied with the Symbolic Solver Environment. See Client User Guide in section 11 for more details about this option.

An example of SOAP request and response messages, sent between client and string-base version of Indefinite Integration Service can be found in Appendix 13.

8.3 TCP monitoring

TCP request/response monitoring can also be useful while examining the service calls. We suggest to use TCP monitor application, provided with Apache Axis distribution. To run it simply call `java org.apache.axis.utils.tcpmon [listenPort targetHost targetPort]`

9 Exposing services to the outside world

9.1 Registration on broker

Each service advertises itself with the Monet Broker. It involves the registration of a service with the Broker Registry Manager [8]. By default it is performed when during service installation by Symbolic Solver Environment installation manager. Furthermore service can be register with the broker independently at any time by calling be registration utility from Symbolic Solver Environment:

```
java RegisterServiceWithBroker [-reg|-dereg|-modify] <broker URL> <service MSDL (file
name or reference)>
```

9.2 Client interfaces

Symbolic Solver Environment distribution provides each symbolic service with three flavours of client interface:

- command line interface
- graphical user interface
- web-based interface

First two interfaces are universal for all services, since they allow the user to change target service and broker URLs, ports, number of service arguments, etc. The third one is specific to each service and is automatically generated by service installation manager of Symbolic Solver Environment.

9.2.1 Command line client

Client-side toolkit of Symbolic Solver offers 3 client java classes to call Monet symbolic services from the command line. The number three is corresponding to number of different encodings for OpenMath objects that each symbolic service supports. So we have three clients to operate with string-based, DOM XML and RIACA encodings of OpenMath.

To call a service using this simple interface user needs to specify service URL and URLs or local file names of required mathematical arguments. For example for string-based version of series expansion service that requires 4 input argument for function, variable, origin and truncation order one can type

```
java SymbolicServiceStrClient
     http://ptibonum.scl.csd.uwo.ca:16661/axis/services/SeriesExpansionService
     http://www.orcca.on.ca/MONET/samples/OpenMath/OM_sin.xml
```

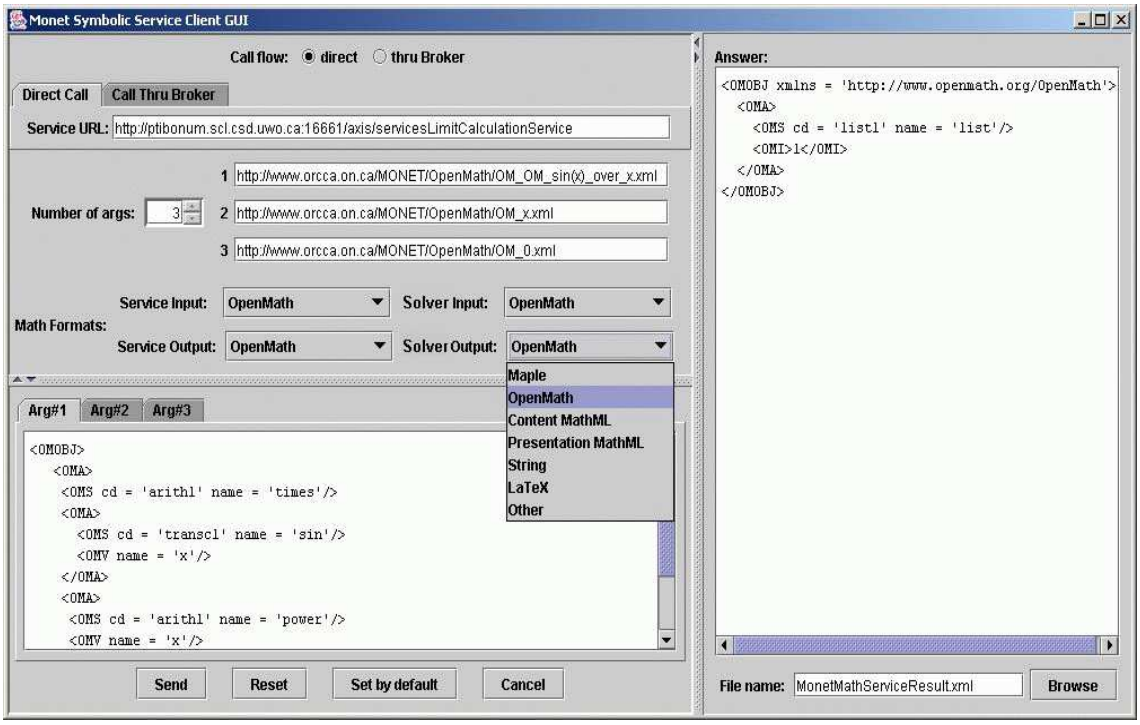

Figure 5: GUI for Symbolic Service Client

```
http://www.orcca.on.ca/MONET/samples/OpenMath/OM_x.xml
http://www.orcca.on.ca/MONET/samples/OpenMath/OM_0.xml
http://www.orcca.on.ca/MONET/samples/OpenMath/OM_9.xml
```

For detailed instructions about usage of command line client interface please refer to the Symbolic Solver User Guide, partly presented in section 11.

9.2.2 GUI

The UWO implementation of Symbolic Solver includes a graphical interface for service clients (figure 5). Likewise command-line version, this interface is universal for all symbolic services, since it allows one to change dynamically the service URL and the number of service arguments.

Among features of the developed GUI application we offer alternative ways to input service arguments: the user may choose to enter them as URL references or local files names or alternatively type their contents in the text editor areas provided. Using drop-down menus

client can easy choose preferred mathematical formats for service input/output, according to 5.2.3. The response from service appears in the right text panel, where it can be browsed, edited and then saved in a file. This interface allows changing a client-service call pattern from "direct" to "via-broker" mode (see User Guide for Clients in section 11).

9.3 Web Client

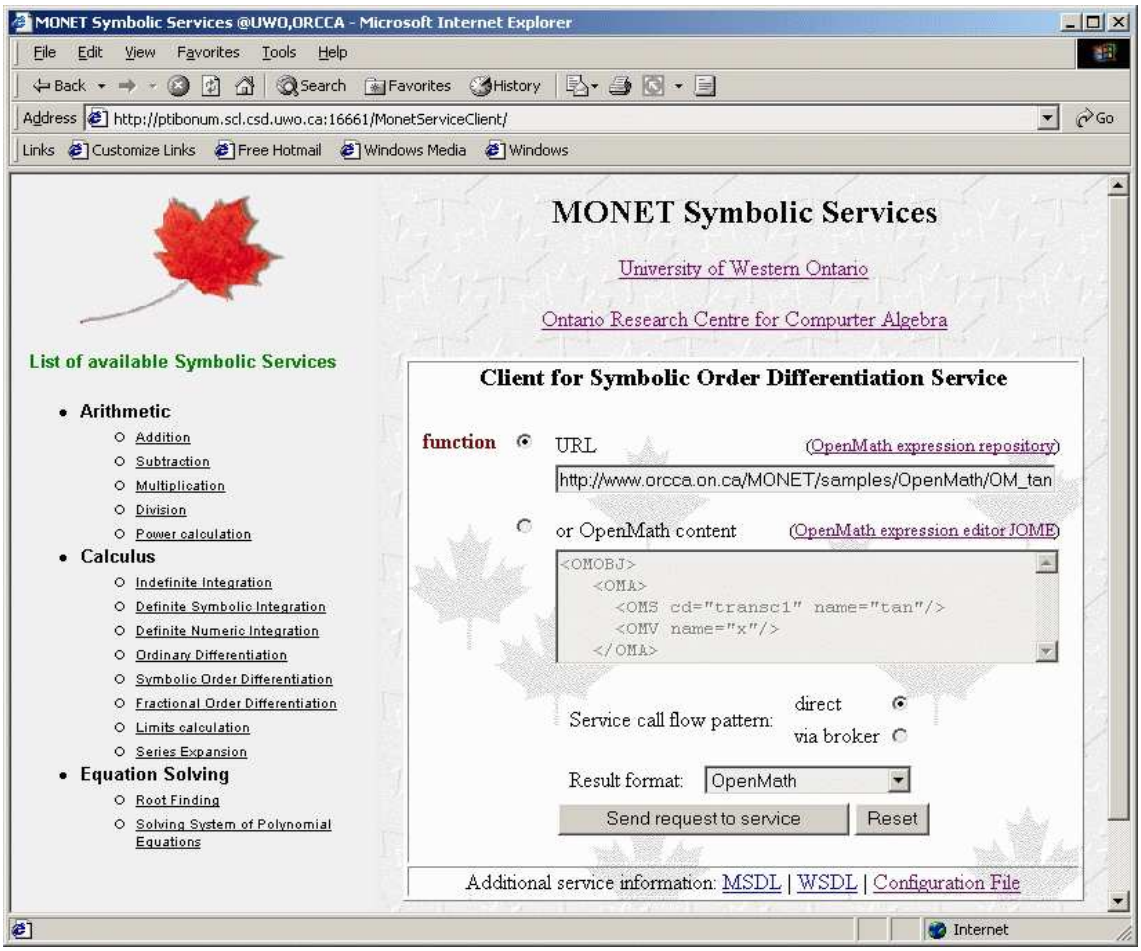

Figure 6: GUI for Symbolic Service Client

Two web interfaces for symbolic service clients are independently offered by both Bath and UWO implementations. They set access to symbolic services through HTML and Java Server Pages [20]. Each developed symbolic service has assigned to it web-client automatically generated. Tools for generation web clients for symbolic services are included in the Symbolic Solver Environment software distribution.

Because those web interfaces are specific to their services, number, order and sometimes type of arguments sent to service can be checked on a client-side, which helps user to formulate a valid request for the service.

The other positive side of having web-client is that users, who need to call symbolic services do not have to have any Symbolic Solver-related software installed on their sides: they just open a corresponding page in the Internet browser and send their requests.

The Bath web clients for deployed services are available at `http://agentcities.cs.bath.ac.uk:8090/axis/maple-service/www/invoke/TSInvoke.jsp` with username `monet` and password `monet` .

The UWO symbolic service web clients are accessible at `http://ptibonum.scl.csd.uwo.ca:16661/MonetServiceClient/`. The screen shot of web page for symbolic order differentiation service is shown in Figure 6.

10 Services Developed

10.1 List of Available Services

The following services are developed and running on the Monet web server at the University of Western Ontario:

- Arithmetical expression simplification services
- Indefinite integration service
- 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)
- Solving systems of polynomials
- Math format conversion

The following services are installed and running on the Monet web server at the University of Bath:

- Simple symbolic integration service
- Gauss Legendre Integration
- Service for evaluation of mathematical expressions in Maple

10.2 Symbolic Service Example: Limit Calculation Service

10.2.1 Mathematical problem description

Input:

- function $f : \mathbb{R} \to \mathbb{R}$
- variable x
- limit point $a \in \mathbb{R}$

Output:

- $L = \lim_{x \to a} f(x)$

Pre-Condition:

- function f has limit at the point a: $\lim_{x \to a+} f(x) = \lim_{x \to a-} f(x)$

Post-Condition:

- $\forall \varepsilon > 0.\, \exists \delta > 0.\, \forall x.\, 0 < |x - a| < \delta \Rightarrow |f(x) - L| < \varepsilon$

The following describes the same problem, using Mathematical Problem Description Ontology [5]

```
<monet:definitions xmlns:monet="http://monet.nag.co.uk/monet/ns"
                   xmlns:om="http://www.openmath.org/OpenMath"
                   targetNamespace="http://monet.nag.co.uk/problems/" >

 <monet:problem name ="limit_calculation">
  <monet:header>
    <monet:taxonomy taxonomy="http://gams.nist.gov" code="GamsO" />
  </monet:header>

  <monet:body>

    <monet:input name="function">
      <monet:signature>
        <om:OMOBJ>
          <om:OMA>
            <om:OMS cd="sts"  name="mapsto" />
            <om:OMV name="R"/>
            <om:OMV name="R"/>
          </om:OMA>
        </om:OMOBJ>
      </monet:signature>
    </monet:input>

    <monet:input name="var">
      <monet:signature>
        <om:OMOBJ>
          <om:OMV name="R"/>
        </om:OMOBJ>
      </monet:signature>
    </monet:input>

    <monet:input name="limit_point">
      <monet:signature>
        <om:OMOBJ>
```

```
      <om:OMV name="R"/>
    </om:OMOBJ>
  </monet:signature>
</monet:input>

<monet:output name="limit">
  <monet:signature>
    <om:OMOBJ>
      <om:OMV name="R"/>
    </om:OMOBJ>
  </monet:signature>
</monet:output>

<monet:pre-condition>
 <OMA>
   <OMS cd="relation1" name="gt" />
   <OMA>
     <OMS cd="limit" name="limit"/>
     <OMV name="a"/>
     <OMS cd="limit" name="below"/>
     <OMBIND>
       <OMS cd="fns1" name="lambda"/>
       <OMBVAR>
         <OMV name="x"/>
       </OMBVAR>
       <OMA>
         <OMV name="f"/>
         <OMV name="x"/>
       </OMA>
     </OMBIND>
   </OMA>
   <OMA>
     <OMS cd="limit" name="limit"/>
     <OMV name="a"/>
     <OMS cd="limit" name="above"/>
     <OMBIND>
       <OMS cd="fns1" name="lambda"/>
       <OMBVAR>
         <OMV name="x"/>
       </OMBVAR>
       <OMA>
         <OMV name="f"/>
         <OMV name="x"/>
       </OMA>
     </OMBIND>
   </OMA>
 </OMA>
</monet:pre-condition>
```

```
<monet:post-condition>
  <OMOBJ>
    <OMBIND>
      <OMS cd="quant1" name="exists" />
      <OMBVAR>
        <OMV name="epsilon" />
      </OMBVAR>
      <OMS cd="logic1" name="implies" />
        <!--  delta > 0  -->
        <OMA>
          <OMS cd="relation1" name="gt" />
          <OMV name="epsilon" />
          <OMI>0</OMI>
        </OMA>
        <!-- exists delta (delta > 0 => forall x |x-a|<delta => |f(x)-L|<epsilon)-->
        <OMBIND>
          <OMS cd="quant1" name="exists" />
          <OMBVAR>
            <OMV name="delta" />
          </OMBVAR>
          <OMA>
          <!-- delta > 0  => forall x |x-a|<delta => |f(x)-L|<epsilon -->
            <OMS cd="logic1" name="implies" />
            <!-- delta > 0  -->
            <OMA>
              <OMS cd="relation1" name="gt" />
              <OMV name="delta" />
              <OMI>0</OMI>
            </OMA>
            <!--  forall x (|x-a|<delta => |f(x)-L|<epsilon)  -->
            <OMBIND>
              <OMS cd="quant1" name="forall" />
              <OMBVAR>
                <OMV name="x" />
              </OMBVAR>
              <!--  |x-a|<delta => |f(x)-L|<epsilon  -->
              <OMA>
                <OMS cd="logic1" name="implies" />
                <OMA>
                  <OMS cd="relation1" name="lt" />
                  <OMA>
                    <OMS cd="arith1" name="abs" />
                    <OMA>
                      <OMS cd="arith1" name="minus" />
                      <OMV name="x" />
                      <OMV name="a" />
                    </OMA>
                  </OMA>
                  <OMV name="delta" />
```

```
                    </OMA>
                    <OMA>
                      <OMS cd="relation1" name="lt" />
                      <OMA>
                        <OMS cd="arith1" name="abs" />
                        <OMA>
                          <OMS cd="arith1" name="minus" />
                          <OMA>
                             <OMV name="f" />
                             <OMV name="x" />
                          </OMA>
                          <OMV name="L" />
                        </OMA>
                      </OMA>
                      <OMV name="epsilon" />
                    </OMA>
                  </OMA>
                </OMBIND>
              </OMA>
            </OMBIND>
          </OMBIND>
        </OMOBJ>
      </monet:post-condition>

    </monet:body>

   </monet:problem>
</monet:definitions>
```

10.2.2 Service configuration file

This file is created by the author of the limit calculation service.

```
<mathServer>

 <msdl>

   <service name ="LimitCalculationService">

   <classification>
     <taxonomy taxonomy="http://gams.nist.gov" code="Gams0"/>
     <problem href="http://monet.nag.co.uk/problems/series_expansion"/>
     <format> http://www.openmath.org/cdfiles/cdgroups/mathml.cdg </format>
     <directive-type href="http://monet.nag.co.uk/owl#evaluate"/>
   </classification>

   <implementation>
     <software name="maple9" href="http://monet.nag.co.uk/owl#Maple9"/>
     <hardware name="ORCCAWebServer" href ="http://monet.nag.co.uk/owl#PentiumSystem"/>
     <action role="execute" name="runService"/>
   </implementation>

  <service-interface-description name="" href =""/>

   <service-binding>
     <map operation ="runService"
          action="invokeService"
          problem-reference="http://monet.nag.co.uk/problems/series_expansion"/>
     <message-construction
          io-ref ="http://monet.nag.co.uk/problems/series_expansion#function"
          message-name ="" message-part =""/>
     <message-construction
          io-ref ="http://monet.nag.co.uk/problems/series_expansion#var"
          message-name ="" message-part =""/>
     <message-construction
          io-ref ="http://monet.nag.co.uk/problems/series_expansion#limit_point"
          message-name ="" message-part =""/>
     <message-construction
          io-ref ="http://monet.nag.co.uk/problems/series_expansion#limit"
          message-name ="" message-part =""/>
   </service-binding>

   <service-metadata/>

  </service>
 </msdl>

 <services>
```

```
  <service name="LimitCalculationService" call="MONET_limit_module:-monet_limit"/>
 </services>

 <implementation language = "maple">
   MONET_limit_module := module()
    export monet_limit;
      monet_limit:=proc(function,var,limit_point);
          limit(function,var=limit_point);
      end:
    end:
 </implementation>
</mathServer>
```

10.2.3 Service MSDL

When service is installed its MSDL gets changed: the service interface description, interface to broker and service binding parts are updated by the Symbolic Solver Environment installation manager.

```
<definitions
targetNamespace="http://www.orcca.on.ca/MONET/samples/msdl"
             xmlns="http://monet.nag.co.uk/monet/ns">

 <service name ="LimitCalculationService">

  <classification>
    <taxonomy taxonomy="http://gams.nist.gov" code="Gams0"/>
    <problem href="http://monet.nag.co.uk/problems/limit_calculation"/>
    <format> http://www.openmath.org/cdfiles/cdgroups/mathml.cdg </format>
    <directive-type href="http://monet.nag.co.uk/owl#evaluate"/>
  </classification>

  <implementation>
    <software name="maple9" href="http://monet.nag.co.uk/owl#Maple9"/>
    <hardware name="ORCCAWebServer" href ="http://monet.nag.co.uk/owl#PentiumSystem"/>
    <action role="execute" name="runService"/>
  </implementation>

  <service-interface-description name="LimitCalculationWSDL"
href="http://ptibonum.scl.csd.uwo.ca:16661/axis/services/LimitCalculationService?wsdl"/>

  <service-binding>
    <map operation ="" action=""
         problem-reference="http://monet.nag.co.uk/problems/limit_calculation"/>
    <message-construction
          io-ref ="http://monet.nag.co.uk/problems/limit_calculation#function"
          message-name ="" message-part =""/>
    <message-construction
          io-ref ="http://monet.nag.co.uk/problems/limit_calculation#var"
          message-name ="" message-part =""/>
    <message-construction
          io-ref ="http://monet.nag.co.uk/problems/limit_calculation#limit_point"
          message-name ="" message-part =""/>
    <message-construction
          io-ref ="http://monet.nag.co.uk/problems/limit_calculation#limit"
          message-name ="" message-part =""/>
  </service-binding>

  <service-metadata/>

  <broker-interface>
    <service-URI>
```

```
      http://ptibonum.scl.csd.uwo.ca:16661/axis/services/LimitCalculationService
    </service-URI>
    <broker-interface-description>
      http://ptibonum.scl.csd.uwo.ca:16661/axis/services/LimitCalculationService?wsdl
    </broker-interface-description>
  </broker-interface>
 </service>
</definitions>
```

10.2.4 Generated java code

```
  / ** Autocreated by Symbolic Solver service installation module **/

import nl.tue.win.riaca.openmath.lang.OMObject; import
org.w3c.dom.Node;

public class LimitCalculationServiceImpl{

  private String configFile  =
      "http://www.orcca.on.ca/MONET/samples/configfiles/limitServiceConfig.xml";
  private String serviceName = "LimitCalculationService";
  private String pathToMaple = "maple";

  public String runService (OMObject[] mathArgs){

     FormatConversionTools fct = getFormatConversionTools();
     try{
         MapleServiceImpl msi =
           new MapleServiceImpl(configFile,pathToMaple,fct,serviceName,mathArgs);
         return msi.runService();
     } catch (Exception e){
          e.printStackTrace();
       return "++ ERROR in MapleServiceImpl.runService: "+e.toString();
     }
  }
//-------------------------------------------------------------------------------------
  public String runService (OMObject[] mathArgs, MathFormats types){

     FormatConversionTools fct = getFormatConversionTools();
     try{
          MapleServiceImpl msi =
            new MapleServiceImpl(configFile,pathToMaple,fct,types,serviceName,mathArgs);
          return msi.runService();
     } catch (Exception e){
          e.printStackTrace();
       return "** ERROR in calling MapleServiceImpl.runService: "+e.toString();
     }
  }
//-------------------------------------------------------------------------------------
  public String  checkService (OMObject[] mathArgs){
     try{
         MapleServiceImpl msi = new MapleServiceImpl(configFile,serviceName, mathArgs);
         return msi.checkService(configFile,serviceName, mathArgs);
     } catch (Exception e){
          e.printStackTrace();
       return "-- ERROR in calling MapleServiceImpl.runService : "+e.toString();
     }
  }
```

```
//-------- String-based OM args ------------------------------------------------------
  public String runService (String[] mathArgs){

     FormatConversionTools fct = getFormatConversionTools();
     try{
         MapleServiceImpl msi =
            new MapleServiceImpl(configFile,pathToMaple,fct,serviceName, mathArgs);
         return msi.runService();
     } catch (Exception e){
          e.printStackTrace();
       return "++ ERROR in MapleServiceImpl.runService: "+e.toString();
     }
  }
//------------------------------------------------------------------------------------
  public String runService (String[] mathArgs, MathFormats types){

     FormatConversionTools fct = getFormatConversionTools();
     try{
          MapleServiceImpl msi =
            new MapleServiceImpl(configFile,pathToMaple,fct,types,serviceName,mathArgs);
          return msi.runService();
     } catch (Exception e){
          e.printStackTrace();
       return "** ERROR in calling MapleServiceImpl.runService: "+e.toString();
     }
  }
//------------------------------------------------------------------------------------
  public String  checkService (String[] mathArgs){
     try{
         MapleServiceImpl msi = new MapleServiceImpl(configFile,serviceName, mathArgs);
         return msi.checkService(configFile,serviceName, mathArgs);
     } catch (Exception e){
          e.printStackTrace();
       return "-- ERROR in calling MapleServiceImpl.runService : "+e.toString();
     }
  }
//-------- XML DOM-based OM args -----------------------------------------------------
  public String runService (Node[] mathArgs){

     FormatConversionTools fct = getFormatConversionTools();
     try{
         MapleServiceImpl msi =
             new MapleServiceImpl(configFile,pathToMaple,fct,serviceName,mathArgs);
         return msi.runService();
     } catch (Exception e){
          e.printStackTrace();
       return "++ ERROR in MapleServiceImpl.runService: "+e.toString();
     }
  }
```

```
//-------------------------------------------------------------------------------------------
  public String runService (Node[] mathArgs, MathFormats types){

     FormatConversionTools fct = getFormatConversionTools();
     try{
          MapleServiceImpl msi =
            new MapleServiceImpl(configFile,pathToMaple,fct,types,serviceName,mathArgs);
          return msi.runService();
     } catch (Exception e){
          e.printStackTrace();
       return "** ERROR in calling MapleServiceImpl.runService: "+e.toString();
     }
  }
//-------------------------------------------------------------------------------------------
  public String  checkService (Node[] mathArgs){
     try{
         MapleServiceImpl msi = new MapleServiceImpl(configFile,serviceName, mathArgs);
         return msi.checkService(configFile,serviceName, mathArgs);
     } catch (Exception e){
          e.printStackTrace();
       return "-- ERROR in calling MapleServiceImpl.runService : "+e.toString();
     }
  }
//-------------------------------------------------------------------------------------------
  public FormatConversionTools getFormatConversionTools(){

     FormatConversionTools fct = new FormatConversionTools();
      fct.setOpenMathToMaplePackage("/MonetSymbolicSolver-UWO//maple/omo_to_mob.mpl");
      fct.setMapleToOpenMathPackage("/MonetSymbolicSolver-UWO//maple/mob_to_omo.mpl");
      fct.setCMathMLToOpenMathXSLT("file:////MonetSymbolicSolver-UWO/xslt/cmmltoom.xsl");
     return fct;
  }
}
```

10.2.5 Generated WSDD files

Deployment descriptor

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
            xmlns:om="http://www.openmath.org/OpenMath">
   <service name="LimitCalculationService" provider="java:RPC">
      <requestFlow>
           <handler type="soapmonitor"/>
      </requestFlow>
      <responseFlow>
           <handler type="soapmonitor"/>
      </responseFlow>
      <parameter name="wsdlTargetNamespace" value="urn:axis.sosnoski.com"/>
      <parameter name="wsdlServiceElement"  value="PersonLookupService"/>

      <parameter name="className" value="LimitCalculationServiceImpl"/>
      <parameter name="allowedMethods" value="runService checkService "/>

      <!--beanMapping for MathFormats and OMObjects -->
      <beanMapping qname="urn:MathFormats"
         xmlns:urn="http://www.orcca.on.ca/MONET"
         languageSpecificType="java:MathFormats"/>
      <beanMapping qname="om:OMOBJ"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMObject"/>
      <beanMapping qname="om:OMS"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMSymbol"/>
      <beanMapping qname="om:OME"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMError"/>
      <beanMapping qname="om:OMSTR"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMString"/>
      <beanMapping qname="om:OMB"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMByteArray"/>
      <beanMapping qname="om:OMBIND"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMBinding"/>
      <beanMapping qname="om:OMV"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMVariable"/>
      <beanMapping qname="om:OMI"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMInteger"/>
      <beanMapping qname="om:OMF"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMFloat"/>
      <beanMapping qname="om:OMATTR"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMAttribution"/>
      <beanMapping qname="om:OMA"
        languageSpecificType="java:nl.tue.win.riaca.openmath.lang.OMApplication"/>
   </service>
</deployment>
```

Undeployment descriptor

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd/">
   <service name="LimitCalculationService"/>
</undeployment>
```

10.2.6 Generated WSDL

Because WSDL generated file for this service is very long, we will include here only few fragments demonstrating the parameters of SOAP request and response messages and service call interface for case of string-based input and output. The full version of this service descriptor is available at `http://ptibonum.scl.csd.uwo.ca:16661/axis/services/LimitCalculationService?wsdl`

```
<?xml version="1.0" encoding="UTF-8"?> <wsdl:definitions
targetNamespace="urn:axis.sosnoski.com"
                  xmlns="http://schemas.xmlsoap.org/wsdl/"
                  [.. snip ..]
                  xmlns:tns1="http://www.openmath.org/OpenMath"
                  xmlns:tns2="http://www.orcca.on.ca/MONET"
                  [.. snip ..]
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
    [.. snip ..]
    <complexType name="MathFormats">
     <sequence>
        <element name="solverInputType" type="xsd:int"/>
        <element name="solverOutputType" type="xsd:int"/>
        <element name="serviceInputType" type="xsd:int"/>
        <element name="serviceOutputType" type="xsd:int"/>
    </sequence>
   </complexType>
   [.. snip ..]
   <complexType name="ArrayOf_xsd_string">
     <complexContent>
       <restriction base="soapenc:Array">
         <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]" />
       </restriction>
     </complexContent>
   </complexType>
 </wsdl:types>

 <wsdl:message name="runServiceRequest">
   <wsdl:part name="in0" type="impl:ArrayOf_xsd_string"/>
 </wsdl:message>
 <wsdl:message name="runServiceResponse">
```

```
   <wsdl:part name="runServiceReturn" type="xsd:string"/>
 </wsdl:message>
 <wsdl:message name="runServiceRequest1">
    <wsdl:part name="in0" type="impl:ArrayOf_xsd_string"/>
    <wsdl:part name="in1" type="tns2:MathFormats"/>
 </wsdl:message>
 [.. snip ..]

 <wsdl:portType name="LimitCalculationServiceImpl">
   <wsdl:operation name="runService" parameterOrder="in0">
     <wsdl:input message="impl:runServiceRequest" name="runServiceRequest"/>
     <wsdl:output message="impl:runServiceResponse" name="runServiceResponse"/>
   </wsdl:operation>
   <wsdl:operation name="runService" parameterOrder="in0 in1">
     <wsdl:input message="impl:runServiceRequest1" name="runServiceRequest1"/>
     <wsdl:output message="impl:runServiceResponse1" name="runServiceResponse"/>
   </wsdl:operation>
 </wsdl:portType>
  [.. snip ..]

 <wsdl:binding name="LimitCalculationServiceSoapBinding"
               type="impl:LimitCalculationServiceImpl">
   <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="runService">
     <wsdlsoap:operation soapAction=""/>
     <wsdl:input name="runServiceRequest">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                      namespace="http://DefaultNamespace" use="encoded"/>
     </wsdl:input>
     <wsdl:output name="runServiceResponse">
       <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                      namespace="urn:axis.sosnoski.com" use="encoded"/>
     </wsdl:output>
   </wsdl:operation>
   [.. snip ..]
 </wsdl:binding>

 <wsdl:service name="PersonLookupService">
   <wsdl:port binding="impl:LimitCalculationServiceSoapBinding"
              name="LimitCalculationService">
     <wsdlsoap:address
      location="http://ptibonum.scl.csd.uwo.ca:16661/axis/services/LimitCalculationService"/>
   </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

10.3 Service Call

To calculate the limit of the function $\frac{\sin x}{x}$ at the point 0 one may use a command-line base client to the limit calculation service by calling the following sequence:

```
java SymbolicServiceStrClient -si 1 -so 1 -SOAP
     http://ptibonum.scl.csd.uwo.ca:16661/axis/services/LimitCalculationService
     http://www.orcca.on.ca/MONET/samples/OpenMath/OM_sin-over_x.xml
     http://www.orcca.on.ca/MONET/samples/OpenMath/OM_x.xml
     http://www.orcca.on.ca/MONET/samples/OpenMath/OM_0.xml
```

Another service invocation is possible by going to `http://ptibonum.scl.csd.uwo.ca:16661/MonetServiceClient/html/LimitCalculationService_client.html` and typing in three arguments:

- **Function:**

```
<OMOBJ>
   <OMA>
    <OMS cd = 'arith1' name = 'times'/>
    <OMA>
      <OMS cd = 'transc1' name = 'sin'/>
      <OMV name = 'x'/>
    </OMA>
    <OMA>
     <OMS cd = 'arith1' name = 'power'/>
     <OMV name = 'x'/>
     <OMI>-1</OMI>
    </OMA>
   </OMA>
</OMOBJ>
```

- **Variable:**

```
<OMOBJ>
   <OMV name="x" />
</OMOBJ>
```

- **Limit point**

```
<OMOBJ>
   <OMI>0</OMI>
</OMOBJ>
```

The service will get the request and return the answer

```
<OMOBJ xmlns = 'http://www.openmath.org/OpenMath'>
  <OMA>
    <OMS cd = 'list1' name = 'list'/>
    <OMI>1</OMI>
  </OMA>
</OMOBJ>
```

The service invocation log for calculation of $\lim_{x \to 0} \frac{\sin x}{x}$ is detailed in the Appendix 12.

11 User Guide

This section contains outlines of the user manual for the UWO Symbolic Solver Environment package. Here we introduce basic operations to be performed for symbolic server and service installation, as well as service invocation.

11.1 Server Installation

1. System setting:
   a) Make sure all requirement software listed in 3.1 are installed and available.
   b) Set the following paths as shell variables:
   - path to Java installation: `JAVA_HOME` ,
   - path to Apache Tomcat working directory: `CATALINA_HOME` ,
   - path to Apache Axis home directory: `AXIS_HOME` ,
   - path to Apache Ant home directory: `ANT_HOME` ,
   - path to mathematical solver installation: `MATHSOLVER_PATH` .
2. Unpack Symbolic Solver distribution archive in a convenient location. It is recommended to set a shell variable to store the path to this directory. By default it will be referred as `MONET_SYMBOLIC_SOLVER_HOME` .
3. Check the port number on which the current version of Tomcat is running.
4. Make sure the Axis servlet is reloadable either automatically or manually. For the first option the file `$CATALINA_HOME/conf/server.xml` has to contain the line `<Context path="/axis" docBase="axis" reloadable="true"/>` . For the second option (more recommended) Tomcat configuration has to define the manager role, assigned at least to one username. For more information about enabling reloading Axis please refer to user manual, section “Reload Axis”.
5. Run `$MONET_SYMBOLIC_SOLVER_HOME/mathserver_init.sh [port] [-reloadaxis|-ra]` Specify argument `port` if different from the default value 8080. If Axis is not set to be autoreloadable, use the key ”-ra” or ”-reloadaxis” to force Axis reload at the end of server installation.
6. To verify that installation of the Symbolic Server was successful go `$AXIS_HOME/WEB_INF/lib` and check presence of the following libraries: `mathServices.jar` , `om-tools.jar` , `om-codecs.jar` , `om-lib.jar` and `risc-openmath-util1.0.3.jar` .

11.2 Installation of New Service

1. Prepare service configuration file, put it in known location on the web or in the local network file system (do not forget to grant reading permission to it).

2. Find out the URL of a Broker on which the service will be registered.

3. Verify the port number on which the current version of Monet Symbolic Server is set up (corresponds to port of Tomcat, on which Symbolic Server is installed in 11.1).

4. Check the full path to the mathematical solver installation if it is not set in the shell variable `MATHSOLVER_PATH` and different from name of the software system in lowcase (for exapmple "axiom", "maple", etc.) This may corrspond to a real path to the software, link or alias.

5. Run
   `cd $MONET_SYMBOLIC_SOLVER_HOME`
   `./mathservice_install.sh <config.file> <broker URL> [port] [path to math solver]`

This script executed will perform the following steps:

a. Parse configuration file and extract service name and service MSDL skeleton.

b. Generate service java core class and put it in `$AXIS_HOME/WEB_INF/lib` directory, so the new service is ready to be deployed on the web server uder Tomcat/Axis. To verify if this step was successful check the presence of the file `<serviceName>Impl.class` in `$AXIS_HOME/WEB_INF/class` .

c. If Axis servlet is not set to be autoreloadable, at this point it will be reloaded by the auxiliary script `reload_axis.sh` .

d. Generate service deployment/undeployment descriptors named `deploy<serviceName>.wsdd` and `undeploy<serviceName>.wsdd` correspondingly. These files will be put in a generated directory `wsdd` under Symbolic Solver home directory `$MONET_SYMBOLIC_SOLVER_HOME`.

e. Deploy the service on the web server using deployment descriptor created in the previous step. After this task is compete the web server is ready to run the new symbolic service as a fully-functional web service.

11.3 Calling Service

Symbolic Services can be called either directly from client or via the Monet Broker. Both of these call flow patterns are supported by all three types of client interface: command-line, GUI and web-based.

- **Call symbolic service directly using command line client**

To call service a symbolic directly using command line client interface, perform the following steps:

1. `cd $MONET_SYMBOLIC_SOLVER_HOME/Client` .
2. If you run the Symbolic Solver Service Client for the first time, run `ant` . It will build java classes and executable jar file from the source.
3. Run the client:
   `cd class`
   `java SymbolicServiceCall [-si n][-mi n][-mo n][-so n][-SOAP] <service URL> <math args>`
   where `service URL` is the URL of calling symbolic service; `math args` mathematical arguments to the service, represented by file names or URL references to files, containing OpenMath objects.

   The following options are available:
   `-SOAP` – use of this key enables tracking of service SOAP request/response.
   `-si` – specifies mathematical format for service input
   `-mi` – specifies mathematical format for mathematical solver input
   `-mo` – specifies mathematical format for mathematical solver output
   `-so` – specifies mathematical format for service output
   `n` – code for mathematical format:
   0: Native syntax of the system of mathematical solver
   1: OpenMath
   2: Content MathML
   3: Presentation MathML
   4: Plain string
   5: LaTeX
   For example, for Maple-based symbolic services the choice of "OpenMath" for all inputs and outputs will lead to conversion between OpenMath and Maple inside of Maple (by calling specially designed Maple packages for this type of conversion).

   The choice of OpenMath format for service input and output together with "Maple" for maple input and "Content MathML" output will invoke another type of call, when OpenMath arguments are converted to Maple syntax *outside* of Maple program by an OpenMath to Maple phrasebook, and output from Maple is converted to Content MathML and then translated to OpenMath by using XSLT stylesheets.

- **Call symbolic service directly using GUI client**

1. `cd $MONET_SYMBOLIC_SOLVER_HOME`
2. Execute `ant -run` .
3. In the window appeared select "direct" item in the "call flow" radio button menu.
4. Enter the service URL in the field provided.
5. Using scroll-field set number of arguments to service call.
6. Enter these arguments either as URL references or local file names in the text fields in upper part of the form or as content of OpenMath in text areas appeared in the bottom of the window.
7. Using drop-down menus choose preferable input and output formats for service request and response, as well as intermediate format of arguments that are passed from service to mathematical solver software.
8. Current setting and parameter values can be saved by pressing on the button "Save as default".
9. Clicking on the button "Reset" will set all fields contents to default values.
10. When you are ready to submit your request to the service, send it by pressing the button "Send".
11. The response from the service will appear in the right-side text area. Its content can be then browsed and saved in a file by using filename text field and "Save" button.

- **Call symbolic service directly using web client**

One can either use existing web clients available at the MONET sites of UWO and Bath or install his own instance of the web client for symbolic services.

1. To access symbolic services at the University of Western Ontario, open in a browser the URL `http://ptibonum.scl.csd.uwo.ca:16661/MonetServiceClient/` ,
2. on the left-side panel choose one of the services offered, then format request to a corresponding service by entering its arguments as URL references or OpenMath content. To help the user, a repository of samples for OpenMath expressions is accessible by the hyper link above each field for service arguments, as well as link to OpenMath editor JOME [22] that helps to get a mathematical expression in the OpenMath format.
3. Set the radio button for service call flow pattern to "direct" value.
4. Send the request by pressing the submit button "Sent request to server".
5. Browse the output from the service on the next page in the field "Result".

- **Call symbolic services via broker**

This option is a temporary solution to provide the client with an interface to symbolic services via Monet Brokers. All client interfaces provided allow to re-direct the call to service via Monet Brokers.

- Command-line client interface offers the following call format:
  `cd $MONET_SYMBOLIC_SOLVER_HOME/Client`
  `java BrokerCallFromClient <Broker URL> [-directcall] <service name> <service args>`
  the option "`-directcall`" allows service to return result direct to the client, when by default the answer from the service is passed to broker first and then to from broker to client.
- To use "via-broker" mode in the GUI client choose the "trough broker" radio button in the upper panel, in the tab opened enter Broker URL and service name. The farther steps are similar to service call in the "direct call" mode.
- To call the service through a broker from a web-based client the user just need to set the service call flow pattern to "via broker" value. The rest of the service calling procedure is equal to the "direct call" case.

12 Invocation Log Example

```
========================================================================
  **  LimitCalculationService invocation log  **
========================================================================

-----------------------
2004-03-29 21:54:32.682
-----------------------
Configuration for mathservice LimitCalculationService has been
loaded from
http://www.orcca.on.ca/MONET/samples/configfiles/limitServiceConfig.xml

-----------------------
2004-03-29 21:54:32.682
-----------------------
Service argument(s): 1: <OMA>
 <OMS cd ='arith1' name ='times'/>
 <OMA>
   <OMS cd ='transc1' name ='sin'/>
   <OMV name ='x'/>
 </OMA>
 <OMA>
  <OMS cd ='arith1' name ='power'/>
  <OMV name ='x'/>
  <OMI>-1 </OMI>
 </OMA>
</OMA> 2: <OMV name ='x'/> 3: <OMI>0 </OMI>

OpenMath encoding format: RIACA


-----------------------
```

```
2004-03-29 21:54:32.713
------------------------
Invoke the service

Solving Engine: MAPLE

Math Formats:
   Service input  : OpenMath
   Solver  input  : OpenMath
   Solver  output : OpenMath
   Service output : OpenMath

Available tools for math format conversion :
   OpenMath to Maple package : /MonetSymbolicSolver-UWO/maple/omo_to_mob.mpl
   Maple to OpenMath package : /MonetSymbolicSolver-UWO/maple/mob_to_omo.mpl
   Content MathML to OpenMath XSLT  : file://MonetSymbolicSolver-UWO/xslt/cmmltoom.xsl

------------------------
2004-03-29 21:54:32.715
------------------------
Maple operation to evaluate:
  MONET_limit_module:-monet_limit

------------------------
2004-03-29 21:54:32.716
------------------------
List of OpenMath args, submitted to Maple:
  ["<OMA><OMS cd='arith1' name='times'/><OMA><OMS cd='transc1' name='sin'/>
    <OMV name='x'/></OMA><OMA><OMS cd='arith1' name='power'/><OMV name='x'/>
    <OMI>-1</OMI></OMA></OMA>",
  "<OMV name='x'/>",
  "<OMI>0</OMI>"]

------------------------
2004-03-29 21:54:32.717
------------------------
MAPLE input file content:

=========================================================
restart;
 # generated temporary filename for Maple program output
 outputFileName:=
  "/scl/packages/jakarta-tomcat-5.0.16/temp/MathServiceOutputs/mapleOut48485.tmp";
 # file containing package for OpenMath to Maple conversion
 OM2MaplepackageFile:= "/MonetSymbolicSolver-UWO/maple/omo_to_mob.mpl";
 # file containing package for Maple to OpenMath conversion
 maple2OMpackageFile:= "/MonetSymbolicSolver-UWO/maple/mob_to_omo.mpl";
 # path to OM<->Maple converter directory
 MobOMpackageDir:= "/MonetSymbolicSolver-UWO/maple";
```

```
 MONET_limit_module := module()
    export monet_limit;
    monet_limit:=proc(function,var,limit_point);
          limit(function,var=limit_point);
    end:
 end:

try
  olddir := currentdir();
  currentdir(MobOMpackageDir);
  # read package for OpenMath to Maple conversion
  read OM2MaplepackageFile;
  # read package with Maple to OpenMath conversion
  read maple2OMpackageFile;
 try
  omArgs:=["<OMA><OMS cd='arith1' name='times'/><OMA><OMS cd='transc1' name='sin'/>
            <OMV name='x'/></OMA><OMA><OMS cd='arith1' name='power'/>
            <OMV name='x'/><OMI>-1</OMI></OMA></OMA>",
            "<OMV name='x'/>",
            "<OMI>0</OMI>"];
  # convert list of OpenMath expressions to sequence of Maple expression
  mapleArgs:=seq(OpenMathToMobConversion:-omtomaple(omArgs[i]), i=1..nops(omArgs));
  output :=My_limit_module:-my_limit(mapleArgs):
 catch :
  errorFlag:=1;
  err:= lastexception;
  errmsg:= StringTools:-FormatMessage( err[2..-1] );
  #ans:= cat("ERROR: ",errmsg);
  ans:= XMLTools:-PrintToString(MobToOpenMathConversion:-mapletoom_error(errmsg));
 finally:
  if (errorFlag<>1) then
      if type(output,list) then
         ans:= MobToOpenMathConversion:-mapletoom(output);
      else
         ans:= MobToOpenMathConversion:-mapletoom([output]);
      end if;
      ans:= XMLTools:-PrintToString(ans);
  end if;

  fd := fopen(outputFileName,WRITE):
  fprintf(fd, "%s",ans):
  fclose(fd):
 end try:
  currentdir(olddir);
catch :
  err:= lastexception;
  errmsg:= StringTools:-FormatMessage( err[2..-1] );
  # ans:= cat("ERROR: ",errmsg);
```

```
  ans:= XMLTools:-PrintToString(MobToOpenMathConversion:-mapletoom_error(errmsg));
  fd := fopen(outputFileName,WRITE):
  fprintf(fd, "%s",ans):
  fclose(fd):
end try: quit;
=========================================================

-----------------------
2004-03-29 21:54:32.727
-----------------------
MAPLE input file stored as
 /scl/packages/jakarta-tomcat-5.0.16/temp/MathServiceInputs/mapleIn48484.tmp

-----------------------
2004-03-29 21:54:32.727
-----------------------
 Path to Mathematical Solver Software is "maple"

-----------------------
2004-03-29 21:54:32.733
-----------------------
Creating file
/scl/packages/jakarta-tomcat-5.0.16/temp/MathServiceInputs/mapleIn48484.tmp_read
containing statement read
"/scl/packages/jakarta-tomcat-5.0.16/temp/MathServiceInputs/mapleIn48484.tmp";

-----------------------
2004-03-29 21:54:32.734
-----------------------
Executing command :
maple /scl/packages/jakarta-tomcat-5.0.16/temp/MathServiceInputs/mapleIn48484.tmp_read

-----------------------
2004-03-29 21:54:43.531
-----------------------
Execution done. Exit value = 0

-----------------------
2004-03-29 21:54:43.534
-----------------------
The answer from MAPLE:

<OMOBJ xmlns = 'http://www.openmath.org/OpenMath'>
  <OMA>
    <OMS cd = 'list1' name = 'list'/>
    <OMI>1</OMI>
  </OMA>
</OMOBJ>
```

13 Service SOAP messages

Here is an example of SOAP request to string-base version of Indefinite Integration Service with a function $\tan(x)$ as an input parameter.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:runService
         soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         xmlns:ns1="http://ptibonum.scl.csd.uwo.ca:16661/axis/services/IndefIntService">
      <arg0 xsi:type="soapenc:Array"
            soapenc:arrayType="xsd:string[2]"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        <item>
         <OMOBJ>
           <OMA>
             <OMS cd="transc1" name="tan"/>
             <OMV name="x"/>
           </OMA>
         </OMOBJ>
        </item>
        <item>
         <OMOBJ>
           <OMV name="x" />
         </OMOBJ>
        </item>
      </arg0>
    </ns1:runService>
  </soapenv:Body>
</soapenv:Envelope>
```

Then the response from the service is

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:xsd="http://www.w.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:runServiceResponse
         soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         xmlns ns1="http://ptibonum.scl.csd.uwo.ca:16661/axis/services/IndefIntService">
     <ns1:runServiceReturn xsi:type="xsd:string">
        <OMOBJ xmlns = 'http://www.openmath.org/OpenMath'>
          <OMA>
            <OMS cd = 'list1' name = 'list'/>
```

```
            <OMA>
               <OMS cd = 'arith1' name = 'times'/>
               <OMI>-1</OMI>
               <OMA>
                  <OMS cd = 'transc1' name = 'ln'/>
                  <OMA>
                     <OMS cd = 'transc1' name = 'cos'/>
                     <OMV name = 'x'/>
                  </OMA>
               </OMA>
            </OMA>
         </OMOBJ>
      </ns1:runServiceReturn>
   </ns1:runServiceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Note: the real SOAP message body contains entities "<" and ">" instead of "<" and ">" in context of OpenMath objects. We replaced these entities to clarify the example look.

References

[1] Horrocks Ian, *MONET Existing and Emerging Technologies*, Deliverable 3, The MONET Consortium, November 2002.

[2] Stephen Buswell,Olga Caprotti and Mike Dewar, *MONET Architecture Overview*, Deliverable 4, The MONET Consortium, December 2002.

[3] Yannis Chicha, James Davenport, David Roberts, *Mathematical Explanation Ontology*, Deliverable 7, The MONET Consortium, January 2004.

[4] Elena Smirnova, Yannis Chicha, Nick Taylor, *Broker Initial Beta Version*, Deliverable 8, The MONET Consortium, August 2003.

[5] Olga Caprotti, David Carlisle, Arjeh M. Cohen, Mike Dewar, *Mathematical Problem Description Ontology*, Deliverable 11, The MONET Consortium, March 2003.

[6] *Mathematical Query Ontology*, Deliverable 13, The MONET Consortium, March 2003.

[7] Stephen Buswell, Olga Caprotti, Mike Dewar, *Mathematical Service Description Language* , Deliverable 14, The MONET Consortium, March 2003.

[8] Walter Barbera-Medina, Stephen Buswell, Yannis Chicha, Marc Gaetano, Julian Padget, Manfred Riem, Daniele Turi, *Broker API*, Deliverable 18, The MONET Consortium, September 2003.

[9] Marc Aird(ed): *Symbolic Service Beta Version*, Deliverable 15, The MONET Consortium, July 2003.

[10] Marc Aird, Walter Barbera, Elena Smirnova, *Symbolic Service Release Candidate*, Deliverable 21, The MONET Consortium, December 2003.

[11] W3C, *Simple Object Access Protocol (SOAP) 1.1*, `http://www.w3.org/TR/SOAP`.

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

[13] W3C, *Web Service Deployment Descriptor (WSDD) 1.1*, `developer.apple.com/documentation/WebObjects/Web\_Services/Web\_Services/chapter\_4\_section_7.html`.

[14] *OpenMath* `http://www.openmath.org/cocoon/openmath//index.html`.

[15] *RIACA OpenMath Library*, `http://www.openmath.org/cocoon/html/openmath_tutorial/projects/om/lib/build/doc/api/overview-summary.html`.

[16] Sun Microsystems Inc, `http://java.sun.com`, 1995-2003.

[17] The Apache Software Foundation, `http://jakarta.apache.org/tomcat`, 1999-2003.

[18] The Apache Software Foundation, `http://httpd.apache.org`, 1996-2003.

[19] The Apache Software Foundation, `http://ws.apache.org/axis`, 1999-2003.

[20] *JavaServer Pages Technology*, `http://java.sun.com/products/jsp/`, 1999-2003.

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

[22] *JOME: Java OpenMath Editor*, `http://mainline.essi.fr/jome/jome-editor-en.html`

[23] *Maple*, `http://www.maplesoft.com`

[24] *Axiom computer algebra system*, `http://www.nongnu.org/axiom/`

[25] *Derive*, `http://www.derive.com`

[26] *Mathematica*, `http://www.wolfram.com/`

[27] *MatLab*, `http://www.mathworks.com/products/matlab/`