Mathematics over the Internet/Web: A Protocol-based Approach

Paul S. Wang* Qingzhao Guo Weidong Liao
Institute for Computational Mathematics
Kent State University
Kent, Ohio 44242, U.S.A.

Abstract *The* Internet Accessible Mathematical Computation *(IAMC) framework aims to make it easy to supply mathematical computing powers over the Internet/Web. The Mathematical Computation Protocol (MCP), the core part in the IAMC framework, enables developers to create interoperable clients and servers easily and independently. Presented are the specification of MCP, and the design and implementation of our Java based MCP protocol library (JMCP).*

Keywords: Internet, Mathematics, IAMC, Java, Protocol, MCP

1 Background

Internet Accessible Mathematical Computation (IAMC) is an emerging research area that promises to bring the power of interactive mathematical computation to the Web and Internet. Conferences and workshops related to IAMC include the *IAMC Workshop* [12], the *MathML Conference* [6], and *Mathematics on the Web*, a special session at the International Congress of Mathematical Software, 2002. For more background and related activities for IAMC, please refer to the IAMC homepage [11], the Proceedings of the IAMC'99 Workshop [12], and the Workshop on *The Future of Mathematical Communication* [13].

In PDPTA'99 and PDPTA'00 we presented a Java-based mathematical encoding package (JMP)[10], and a prototyping IAMC system [4], respectively. A recent research focus at the Institute for Computational Mathematics (ICM/Kent) has been on the design and implementation of a distributed IAMC framework [7] to make mathematical computation easily and widely available on the Internet/Web. The *Mathematical Computation Protocol* is a centerpiece in the IAMC framework.

*Work reported herein has been supported in part by the National Science Foundation under Grant CCR-0201772 and in part by the Ohio Board of Regents Computer Science Enhancement Funds.

IAMC and MCP

The IAMC framework [7] is designed to make mathematical computing easily accessible and usable on the Web/Internet [8]. Figure 1 shows the overall IAMC framework architecture.

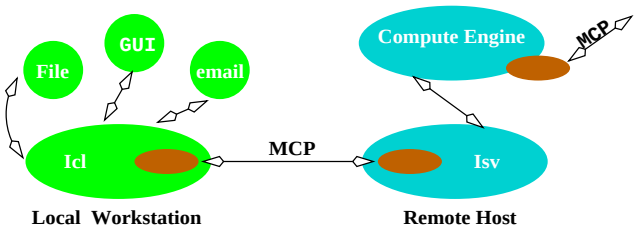

Figure 1: IAMC Architecture

IAMC clients and servers are connected by the Mathematical Computation Protocol (Section 2). MCP aims to be a simple and effective request-response protocol to support one-time transactions and interactive sessions.

Standard and user-defined mathematical data encodings can be used. MCP supports automatic negotiation between the IAMC clients (Icl) and servers (Isv) at the beginning of a computation session to determine the encoding(s) used.

MCP allows the user to interrupt and abort

a computation that is taking too long or becomes unnecessary. A client-generated interrupt can, for example, be delivered by TCP "out-of-band data" to the Isv which will cause the current asynchronous computation to abort, if not already complete.

A Web page containing formulas or equations can also make the mathematical expressions come alive through server-side programming (e.g. CGI, PHP, Servlet) that taps into local or remote IAMC servers. The MCP protocol is designed to also support such applications. Library modules can be developed to enable Web server-side programs to function as compact IAMC clients and easily request and obtain mathematical results from IAMC servers.

2 MCP Protocol

Important characteristics of MCP include:

- Handling one-time computation requests and persistent computation sessions.
- Placing no restrictions on content types or mathematical data representations.
- Supporting the special needs of mathematical computations.
- Permitting both server and client to send requests and to return responses.
- Providing for both synchronous and asynchronous message exchanges.
- Distinguishing protocol control from computation commands.
- Being simple, effective, and extensible.

As described in its original design [8], MCP uses HTTP-style requests and responses to support mathematical computation sessions. Each MCP message consists of a *control line*, a *header*, and an optional *body*. Each header entry is a *key-value pair* on one line terminated by a NEWLINE or a CR-NEWLINE. The header and body are separated by an empty line. The first line of an MCP message is a control line that specifies the *message type*, the *message class*, and a sequence number. The message control line takes one of two forms:

`Request` *Class seqNo*

or

`Response` *Class seqNo statusCode* [*statusString*].

Specific MCP requests belong to different classes that contain request methods to support well-defined operations. An MCP request identifies the request class and a sequence number. The request sequence number *seqNo* is generated by the request originator. A response indicates the class and *seqNo* to identify the request to which it is responding.

A *statusCode* element must be included in the Response message control line, indicating the understand and satisfy of the request. The optional *statusString* is a description of the *statusCode*. For example, 200 OK in the response to a computation request indicates that the request has succeeded, and the result has been returned in the response message.

More detailed information about MCP and MCP protocol specification are available in the ICM technical report Web site[9].

MCP Message Classes

MCP messages are categorized into message classes. The message class limits the available methods, while the class and the method together determines what other header fields and any message body to be given. The method to be used is set in the `Method` header field. The other header fields and the message body can be considered arguments to the method.

Initialization

The `Initialization` class supports session creation and configuration right after client-server connection. A sample `Initialization` request from a client to a server is:

```
Request Initialization C4
Method: canDo

Response C4 200 OK
Server-Name: PolyFactor
Content-Length: 256

"Calculus"=integrate diff taylor...
"Linear Algebra"=vector matrix...
"Complex Analysis"=absValue conjugate
                   realPart ...
...
```

where the client gets the computation capability of the server. The `canDo` format

"*Area_name*"=*command1 command2* ...

gives IAMC servers the freedom to name the areas and commands it supports. The `canDo` data can be collected by an IAMC search engine to make locating IAMC servers easy.

Other `Initialization` methods include: `connect`, `version-negotiation`, `setup`, `canDo`, `content-negotiation`, `security-negotiation`, `dictionary-map`, `end-initialization`.

Computation

Computation is a class of server-side operations to perform application supported computations. The following is a sample message in this class category.

```
Request Computation C7
Method: command
Content-Type: Application/X-Math-MathML
Content-Length: 128
Send-Result: No
Mode: asynchronous

(the message body is the MathML encoding
of p:=2*x^2-y)

Response Computation C7 200 OK
```

The encoding type of the computation command, such as infix, MathML [1], MP [3] is indicated by "Content-Type" header field. MCP aims to support any mathematical encoding. The computation mode can be either synchronous, whereas client will wait until the response is returned, or asynchronous, whereas request returns immediately and any response will be delivered when available.

Control

The control class supplies MCP protocol control and management methods for both the client and server side.

An example of the control request is to abort an unfinished computation. In the asynchronous mode, sometimes the client wants to interrupt a time-consuming computation. This can be done by sending a Control message with method `abort`. Other initialization methods include: `disconnect`, `interrupt`, `reconnect` .

Dialog

The dialog class specifies a class of methods to solicit information from the end user. This is useful when the server needs to ask for additional information in order to complete processing a request. For example, a computation request `integrate(1/x,x,a,b)` will initiate a dialog message with content `Is b, negative, positive or zero`.

3 Design of the MCP Library

A Java class library, JMCP, has been designed and implemented as a reference implementation for the MCP protocol. JMCP makes it easy for developers to create Java-based MCP applications. The reference implementation helps to refine the protocol specification and we hope it can also guide other implementations in Java or other languages.

The following we will discuss some major interfaces and classes in our JMCP library.

1. `McpLink` represents the persistent connection between the server and the client. There should be an object of this class on each side of the client and server. Some methods in an *McpLink* object are:

- `public McpLink(McpURL url)` – create an *McpLink* object using the given *McpURL*.
- `public boolean isAlive()` - request the status of its peer.
- `public void terminate()` - disconnects from its peer.
- `public Command createCommand()` - create a new computational command to be executed through this link.
- `public CanDoList getCanDoList()` - get the computation capability list of the server. This method is used by IAMC clients only.

2. `ResponseHandler` is an interface for synchronous or asynchronous result handling. In concept, the result handling is similar to the Delegation Event Model used in Java GUI event handling. Any objects implementing ResponseHandler could be registered to handle the response to a specific message. When a `ResponseHandler` is registered to a `Message`, the handler's `processResponse()` method will be invoked when the response from the peer side is returned.

   Only one method is specified in this interface.

   - `public void processResponse (Message source)`: Invoked when the response for the message is returned.

3. `Message` is an interface that all the messages transferred over MCP connections, such as `Command, Help, Dialog, and Template` have to implement. The most important method specified in this interface is `addResponseHandler` that is used to handle any response for an MCP message.

   - `public void addResponseHandler (ResponseHandler rhandler)` – register a result handler for the created command.

4. `Command` represents a computational command to be executed in the server side. Some methods in a Command object are:

   - `public void setEncoding(String encoding)` - set the encoding type of the mathematical command.
   - `public String getEncoding()` - get the encoding type of the mathematical computational command.
   - `public void setContent (byte[] content)` - set the content of the command.
   - `public byte[] getContent()` - get the content of the command.
   - `public void execute()` - issue the command for execution in the server side. A registered result handler will take care of receiving the result back from the server side.

5. `Help` represents the help information that the client asks for from the server. The methods in a *Help* object include:

   - `public void get(Command com)` - get help information about a certain command from the server side.
   - `public void get(Topic topic)` - get help information about the specified topic.

6. `Dialog` represents the request information from the server, soliciting additional information about a command request.

7. `McpURLConnection` is the high-level abstraction for MCPLink, usually used by one-time computations.

4 Programming with JMCP

JMCP can be loaded by either an Icl or an Isv. The MCP layer on the server side interacts with the MCP layer on the client side to perform communication.

4.1 McpLink and McpURLConnection

MCP provides both persistent sessions and one-time computations. Correspondingly, JMCP supports two types of connection:

- McpLink — a persistent connection between the server and the client. Initialization messages, such as versionNegotiation will be exchanged right after the connection is built.
- `McpURLConnection` — a high-level connection built over MCPLink. Each McpURLConnection instance is used to make a single request but the underlying MCP Link may be transparently shared by other instances.

These two different connections are shown in Figure 2. We anticipate more flexibility and higher efficiency by providing different connection options. For example, a web based MCP application may use McpURLConnection to gain MCP access to IAMC servers for simplicity, and a standalone MCP application could build a persistent MCP connection for continuous transmission of mathematical data. Moreover, sharing low-level McpLink among multiple McpURLConnection instances may also increase the efficiency of McpLink.

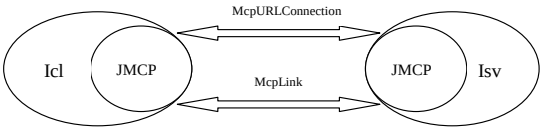

Figure 2: MCP Connections

4.2 Asynchronous MCP Message Handling

JMCP uses the Observer [2] design pattern for asynchronous message delivery and handling, which is also the pattern used in Java delegation-based event handling. Asynchronous message delivery and handling means when a message is submitted to the peer side, the JMCP application does not have to wait for response. In contrast, it will assign a `ResponseHandler` to take care of receiving and processing the result back from the peer side. This strategy is very useful for the long-duration mathematical computation.

The `ResponseHandler` and `Message` interfaces in JMCP package are the starting point for our asynchronous message handling. The `ResponseHandler` is an observer object that is ready to receive and process the response for a message that has been sent to the peer side. Its only method, `processResponse()`, will be invoked when the response is returned. The `Message` interface is a tagging interface that the MCP messages of different types must extend/implement. The `Message` interface provides methods to register/remove `ResponseHandler`.

Using `ResponseHandler` to handle the response for a message follows a three-step procedure. First, a class has to be defined to implement the `ResponseHandler` interface, which includes implementing the details for receiving and processing the potential responses for messages of a specific type. Second, an instance of `ResponseHandler` interface has to be created based on the class implementing this interface. Third, the handler has to be added to the message that is ready to be sent to the peer side.

4.3 A Sample MCP Application

Now let us look at a typical scenario for an application to perform computations following the MCP specification. We will see how an MCP client application is constructed using JMCP to perform a computation. Suppose a client application wants to perform the computation *integrate(sin(x),x)* by employing JMCP class library. The following is the sample code:

```
1) McpURL url = new McpURL("ox.cs.kent
       .edu",9089);
2) McpLink link = new McpLink(url);

3) CanDoList list = link.getCanDoList();
```

```
4) Command comm = link.createCommand();

5) comm.setEncoding(String("Text/Infix");
6) String str = new String(("integrate
   (sin(x),x)");
7) byte[] content = str.getBytes();
8) comm.setContent(content);

9) CommandResponseHandler handler = new
   ResponseHandler();
10) comm.addResponseHandler(handler);

11) comm.execute();
```

First, a connection to the server is established, based on the `McpURL` provided (line 1-2). Based on the established McpLink, the computational capability could be fetched from the server (line 3). To perform a computation, a new command is created(line 4) and the intended encoding and content of the command (line 5-8) are to be specified. We then register a response handler for the command to be executed(line 9-10). After the command (line 11) is issued for execution, the handler registered will retrieve and process the result.

The server side has a symmetric structure, which we will not describe here due to the space limitation.

5 Future Work

The MCP effort investigates ways to establish an application level connection between the client and server in the IAMC Framework, thus making distributed mathematical computation easier. JMCP has been integrated in the latest version of Dragonfly [5] and Starfish [7], which are our prototyping IAMC client and server.

The performance of JMCP is to be evaluated and compared with other protocols, such as HTTP. Authentication and security will also be considered, in the protocol specification and JMCP implementation.

References

[1] R. Ausbrooks, S. Buswell, S. Dalmas, S. Devitt, A. Diaz, R. Hunter, B. Smith, N. Soiffer, R. Sutor, S. Watt. Mathematical Markup Language (MathML) v.2.0. `http://www.w3.org/TR/2000/CR-MathML2-20001113`, Nov. 2000.

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Element of Reusable Object-Oriented Software Published by Addison-Wesley. pp293-299. Dec. 1998.

[3] S. Gray, N. Kajler and P. S. Wang. Design and Implementation of MP, a Protocol for Efficient Exchange of Mathematical Expressions. *J. of Symbolic Computation*, 25(2):213–238, Feb. 1998.

[4] W. Liao and P. S. Wang. Building IAMC: A Layered Approach. In *Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'00)*, pages 1509–1516. Csrea Press, 2000.

[5] W. Liao and P. S. Wang. Dragonfly: A Java-based IAMC Client Prototype. In. *4th Asian Symp. on Computer Mathematics (ASCM'2000)*, 2000.

[6] MathML Intl. Conf. 2000, UIUC Illinois USA, `http://www.mathmlconference.org`, Oct. 20-21, 2000.

[7] P. S. Wang, S. Gray, N. Kajiler, D. Lin, W. Liao, X. Zou. IAMC Architecture and Prototyping: A Progress Report, Proceedings of ISSAC 2001, International Symposium on Symbolic and Algebraic Computation, pp. 337-344, July, 2001.

[8] P. S. Wang. Design and Protocol for Internet Accessible Mathematical Computation. In. *ISSAC'99*, pages 291–298. ACM Press, 1999.

[9] P. S. Wang, S. Gray, Q. Guo, W. Liao. MCP Specification, ICM Technical Report. 2002. `http://icm.mcs.kent.edu/reports/index.html`.

[10] S. Gray, L. Tong, P. S. Wang. The MP Encoding for Distributed Mathematical Computations: An Object-oriented Design and Implementation. In *Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 2084–2090. Csrea Press, 1999.

[11] `http://icm.mcs.kent.edu/research/iamc` (IAMC homepage), `http://icm.mcs.kent.edu/research/iamcproject.html` (IAMC project homepage).

[12] `http://icm.mcs.kent.edu/research/iamc01proceedings.html`, *Proc. of the IAMC'01*, July 2001.

[13] `http://msri.org/calendar/workshops/9900/`. *Proc. of The Future of Mathematical Communication*, Dec. 1999.