IAMC: Internet Accessible Mathematical Computation

Paul S. Wang*
pwang@mcs.kent.edu
http://icm.mcs.kent.edu/~pwang
Institute for Computational Mathematics
Kent State University
Kent, Ohio 44242-0001

April 6, 1998

Abstract

IAMC aims to make mathematical computation services easily accessible on the Internet. The IAMC design leverages existing technologies and makes it possible to access services through TCP/IP, the Web, or email. The overall IAMC architecture, the object-oriented design of a prototype IAMC client and server, as well as the initial specification of the *Mathematical Computation Protocol* which connects IAMC clients and servers are presented. Experience gained with implementation experiments are also reported.

1 Introduction

"The network is the computer" is a concept with global impact. The Internet and the World-wide Web make many kinds of information and services easily accessible. But sharing of mathematical/scientific procedures and results is lagging behind. The *Internet Accessible Mathematical Computing* (IAMC) project investigates how mathematical computations can be made as easily accessible as a Web page.

Ad-hoc methods have been used to make mathematical computing available on the Internet. For example, an extensive table of integrals is network-accessible at the University of California, Berkeley [4]. Also, live system demos are accessible on `SymbolicNet`

*Work reported herein has been supported in part by the National Science Foundation under Grant CCR-9503650

(`http://SymbolicNet.mcs.kent.edu`) either through simple CGI programming or by accessing remote X servers. These approaches are not general, hard to deploy widely, and lacking interoperability.

The importance of technical/mathematical communication on the Internet is underscored by the recent activities at the W3 consortium and elsewhere to make publishing mathematical materials on the Web easy. The MathML working draft[1] defines a SGML language for markup of mathematical expressions. Both presentation (display) and content (computation semantics) markup are supported. The IBM digital publishing group has released the experimental `techexplorer`, a Web browser plug-in that dynamically formats and displays documents containing scientific and mathematical expressions coded in TEX/LATEX. Some MathML is also supported. The `WebEQ` from Geometry Technologies Inc.[2] can display `WebTeX` and a set of MathML by converting them to LATEX images. The W3 consortium's `Amaya` Web browser[3] now demonstrates a prototype implementation of MathML which allows users to browse and edit web pages containing mathematical expressions. Like the rest of the document, these expressions are manipulated through a WYSIWYG interface.

In addition to publishing mathematical content in a Web page, mathematical computation services should be accessible on the Internet directly. These services should comply with a common standard and be mutually compatible for interoperability. Accessing a scientific computing service should be as simple as entering a command, accessing a Web page, or sending email. The IAMC project aims to design and prototype a distributed system that can

- make math-oriented data and services easily and widely accessible on the Internet – directly, via the Web, and by email
- support interactive use of remote *compute servers* almost like local programs
- perform effective and efficient communication of mathematical data over the Internet
- allow exchange and further processing of computational results among heterogeneous compute servers

The overall architecture, designs for the IAMC client and server, and the *Mathematical Computation Protocol* that connects them will be presented. Implementation experiments on a simple stream socket based system are also described.

2 Requirements and General Descriptions

The goal of IAMC is to make accessing and supplying a wide range of mathematically oriented computing services on the Internet simple and easy. Such services include

[1] `http://www.w3.org/TR/WD-math-980106/`
[2] `http://www.webeq.com/webeq/`
[3] `http://www.w3.org/Amaya/`

- Use of remote mathematical computation engines
- Access to remote scientific databases
- Making parallel/super computing accessible
- Access to research software maintained by expert groups around the world
- Distance learning
- Computing via NetPC for high school or occasional users
- Establishing Problem Solving Environments (PSE)

IAMC should be convenient, simple to use, and easy to learn. It must also leverage appropriate existing technologies to reduce R&D and to increase the chance of wide acceptance:

- Internet and the Web
- MathML – the mathematical markup language [9], an extension to HTML [10]
- MP – a binary math data encoding/transfer protocol developed at ICM/Kent
- OpenMath – evolving mathematical data representation and semantics specifications developed by the European Maple group and collaborators [1], [3]
- Java – platform independent programming system with built-in network and GUI support

Brief introductions to MathML and MP can be found in Sections 7.1 and 7.2.

Basically, each IAMC server provides a specific service and has an URL in the form

`iamc://`*hostname:port*`/`*server-id*

During development, we suggest using port 4711 for the per-host IAMC daemon which will invoke specific local IAMC servers according to the *server-id* (Fig. 1). An end user accesses IAMC through an IAMC client. The client may connect to any given server and obtain interactive access to the service. The client can also request computation via email and display/process replies received. IAMC client-server communication uses the *Mathematical Computation Protocol* (MCP).

IAMC services should be available in different ways. A user can invoke an IAMC client directly to connect to any server giving its URL. IAMC service providers can publish and make their services available from Web documents with a hyperlink to an IAMC server. Of course this assumes the Web browser will launch an IAMC client automatically. For simple mathematical services, such as integral table look-up, a HTML form coupled with a CGI program that front-ends an IAMC server can be employed. An IAMC server can either perform the computational tasks directly or use a separate compute engine for the purpose.

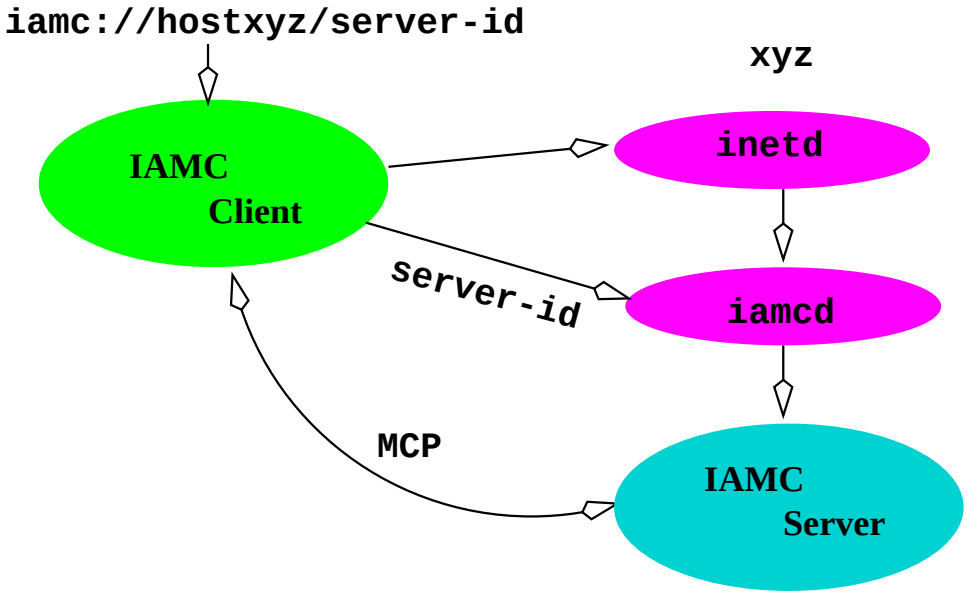

Figure 1: Connecting to An IAMC Server

In addition to using a direct TCP/IP connection, an IAMC client and server can transact business through persistent HTTP [5], regular HTTP/CGI, or even email. An IAMC client may also connect to multiple servers at the same time.

For example, an email-based IAMC may use:

- Request format:

  ```
  Subject: IAMC REQUEST version
  Content-Type: application/x-mcp
  Transfer-Encoding: base64
  To: serverid.IAMC@host
  ```

- Response format:

  ```
  Subject: IAMC RESPONSE status
  Content-Type: application/x-mcp
  Transfer-Encoding: base64
  To: reply address
  ```

The scheme can easily be supported by an autonomous mail processor such as **procmail**.

A CGI-based IAMC may

- use the HTTP POST query
- specify `Content-Type: application/x-mcp`
- send body of query/response without encoding

3 Architecture Overview

The IAMC system consists of the following components:

- IAMC client — The user agent of IAMC.
- IAMC daemon (`iamcd`) — The launch agent for servers. It listens for a stream socket connection at a prescribed port (4711 suggested) and starts requested IAMC servers on demand. The IAMC daemon can also be started by `inetd`.
- IAMC server — Each server provides a specific computational service. An IAMC server can be launched by the `iamcd`, a CGI program, or an autonomous mail processor such as `procmail`.
- MCP — The *Mathematical Computation Protocol* links IAMC clients and servers. A reference implementation of MCP will be written in Java. The MCP classes can simply be linked in a Java-coded IAMC client or server.
- Mathematical Data Encodings — MCP allows flexible use of different mathematical data encoding formats. MP, the *multi protocol* will be the primary encoding format for IAMC. But other formats such as MathML or OpenMath can be used just as well.
- Mathematical expression viewer/editor — IAMC can employ external math-expression viewers/editors as plug-ins. MathML and TEX/LATEX viewer/editors are being actively developed in a number of different places.
- Compute engines — Existing and newly developed systems perform a wide range of technical, scientific, and mathematical computations.

Figure 2 shows an overview of the proposed IAMC architecture.

As an experimental prototype, we will design and implement an IAMC client and server. The programs will be coded in Java to run on any platform with a Java interpreter. The current design of the client and server will be described.

4 Client Design

The IAMC client is the user front end of the system. It interfaces to the user on the one end and to the IAMC server on the other end. A sophisticated IAMC client may offer many functionalities and conveniences. Every IAMC client should have the following responsibilities.

- Provides a GUI
- Connects to specified remote servers
- Receives input from a user (infix, menus, buttons) and/or file

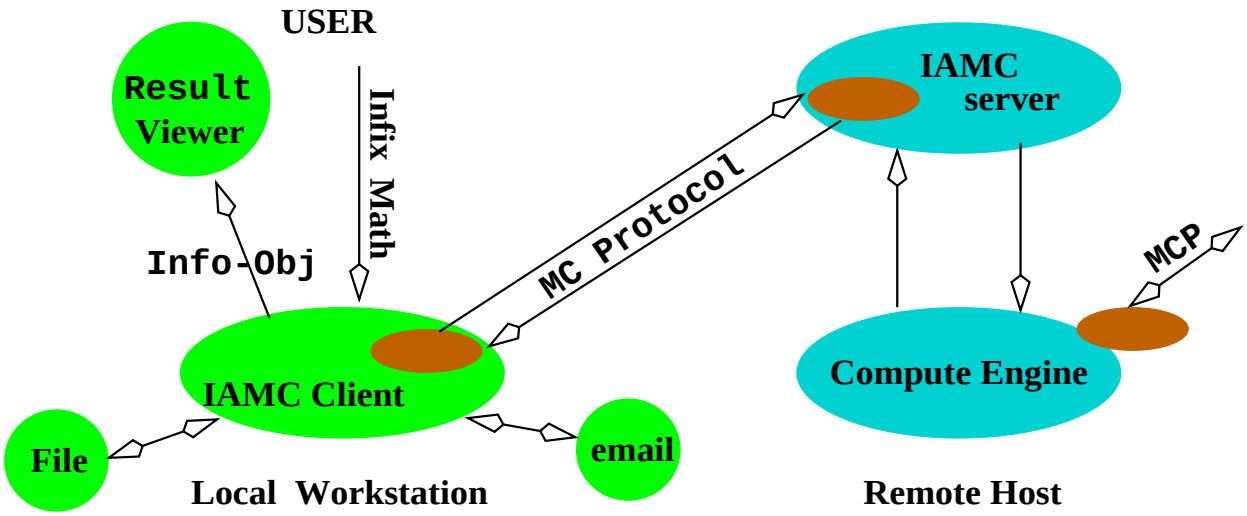

Figure 2: IAMC Architecture Overview

- Edits input, saves input into files
- Parses input, converts input to internal representation, sends input to remove server
- Receives results from remote server
- Controls and manages transactions and sessions
- Produces displayable results: MathML, point sets, GIF, URL, HTML, infix, prefix, and LaTeX.
- Requests and displays server-supplied help/documentation information to the user
- Keeps track of user input and history
- Stores-retrieves files from local file system
- Communicates with remote server in `MCP`
- Provides email interface
- Allows and interfaces to external plug-ins

Figure 4 shows the object-oriented design of a prototype IAMC client. Top-level Java classes are shown.

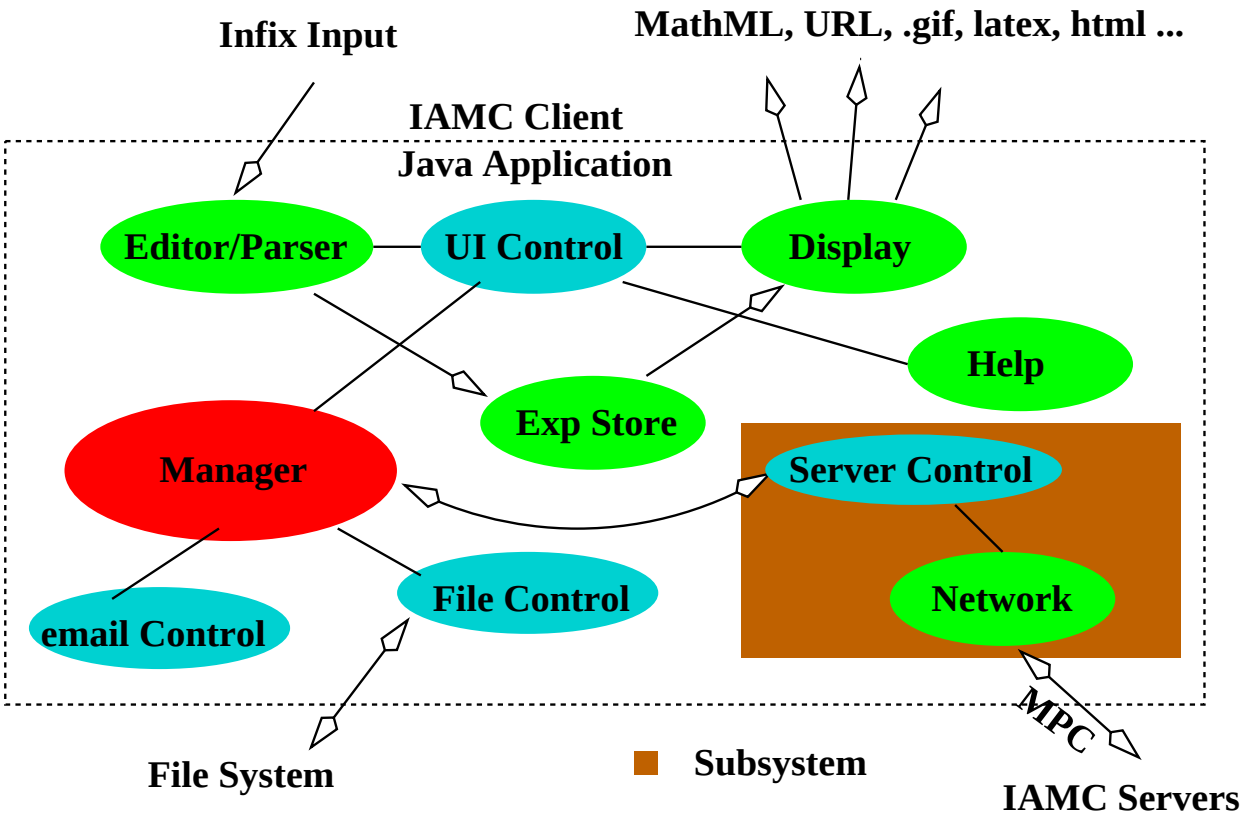

Figure 3: IAMC Client Design

5 Server Design

Each new IAMC service requires a new IAMC server. A server may be spawned by `iamcd`, `cgi`, or an email processor. The client-server communication and the computation control parts of the server will be generic and reusable. Service specific functionalities are supplied either directly in the server or indirectly through an external compute engine. The IAMC server will be responsible for these tasks.

- Receives client requests (from standard input)
- Sends response to client (via standard output)
- Performs requested operation or computation either internally or using an external compute engine
- Controls and interfaces to external compute engine, if so configured
- Provides user help and usage documentation through client
- Keeps log of operations, errors, and statistics

- Understands and speaks MCP

Figure 4 shows the object-oriented design of an IAMC server.

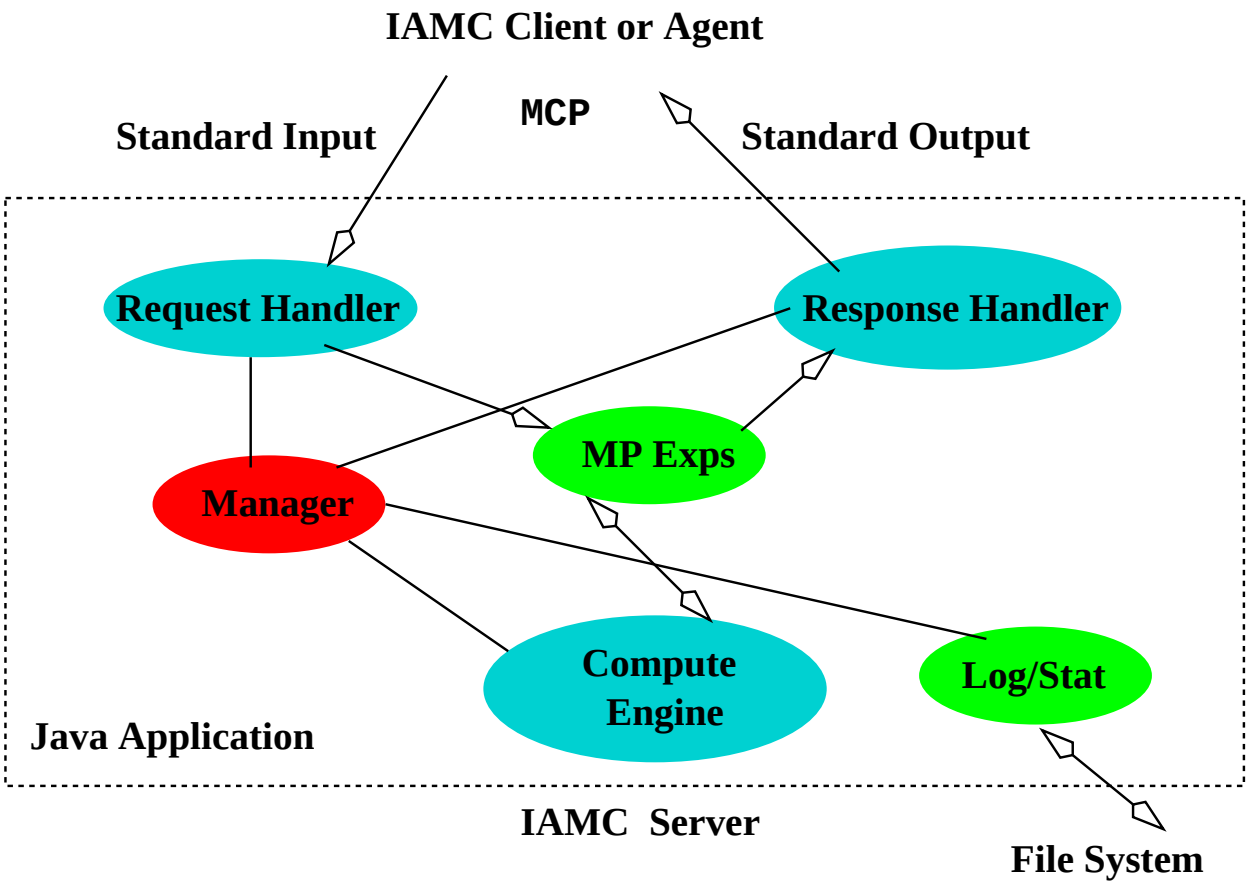

Figure 4: IAMC Server Design

5.1 Server Interface to External Engine

For complicated computations, it is typical to build a stand-alone compute engine. An IAMC server can then be used to adapt such an engine for IAMC service and therefore make the computation widely available on the Internet.

External engine interface issues must be considered. The assumption is that the server and the compute engine are running on the same host computer and they can talk to each other through local interprocess communication. Basically the IAMC server receives requests encoded in a standard format (e.g. MP or MathML). The server must convert the data to a suitable format before sending it to the compute engine (Fig. 5). Mathematical results produced by the compute engine must be converted to a standard format before being returned to the client. Output from the engine may also be in other well-known MIME [2]

types which can be sent along without conversion. For example, the engine can return a GIF image, some HTML documentation, help information, or a URL.

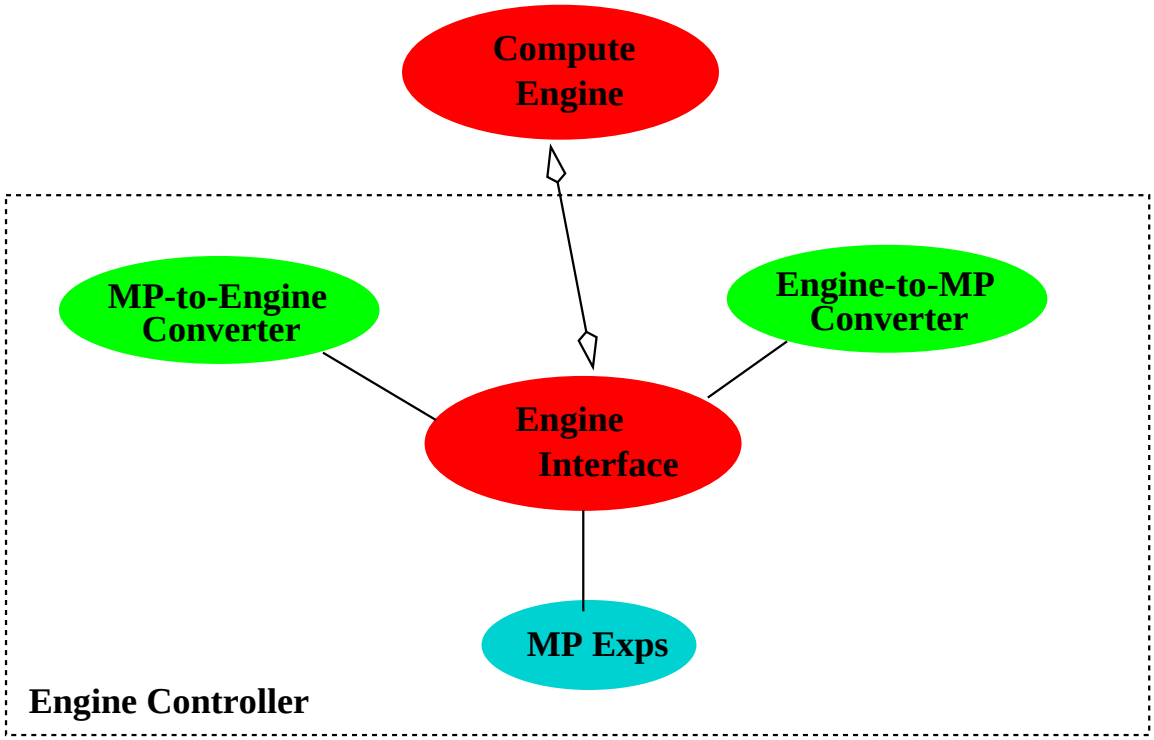

Figure 5: Server Interface to Compute Engine

Other considerations include handling of dialog and error from the external engine and sending interrupts to abort a computation.

6 MCP Protocol Design

6.1 MCP Issues and Considerations

For remote access and control of technical and mathematical computations, a protocol is needed to connect clients and servers. MCP should be a simple, powerful, and flexible protocol whose primary purpose is to support IAMC. But, it may find other uses. Here are some important overriding considerations for MCP:

- Meeting client-to-server and server-to-client requirements
- Using different types of data-transfer encodings
- Employing two-way, sequenced, reliable connection between client and server (TCP/IP, pipe, or CGI)

- Assuming peer-to-peer interactions (one-to-one)

Let's consider client and server requirements separately. Broadly, we distinguish between two types of client requests: computation requests and control requests. For example, *produce the first derivative* is a computation request while *create log file* or *report computation status* is a control request. An IAMC client needs to

- Send computing requests to server
- Receive computational results in various formats from server
- Send control requests to server
- Receive control responses (e.g., server status information)
- Send client status to server
- Receive requests from server
- Respond to requests from server
- Interrupt on-going computation on server
- Disconnect from server

Symmetrically, an IAMC server needs to

- Receive computing requests from client
- Send computational results, in various formats to client
- Send control requests to client
- Receive control responses (e.g., client status information)
- Send server status to client
- Respond to control requests from client
- Disconnect from client

It is also possible to apply MCP to connect multiple compute engines together, on a one-to-one basis, in a *Problem Solving Environment* (Fig. 6) where a server can be a client and vise versa.

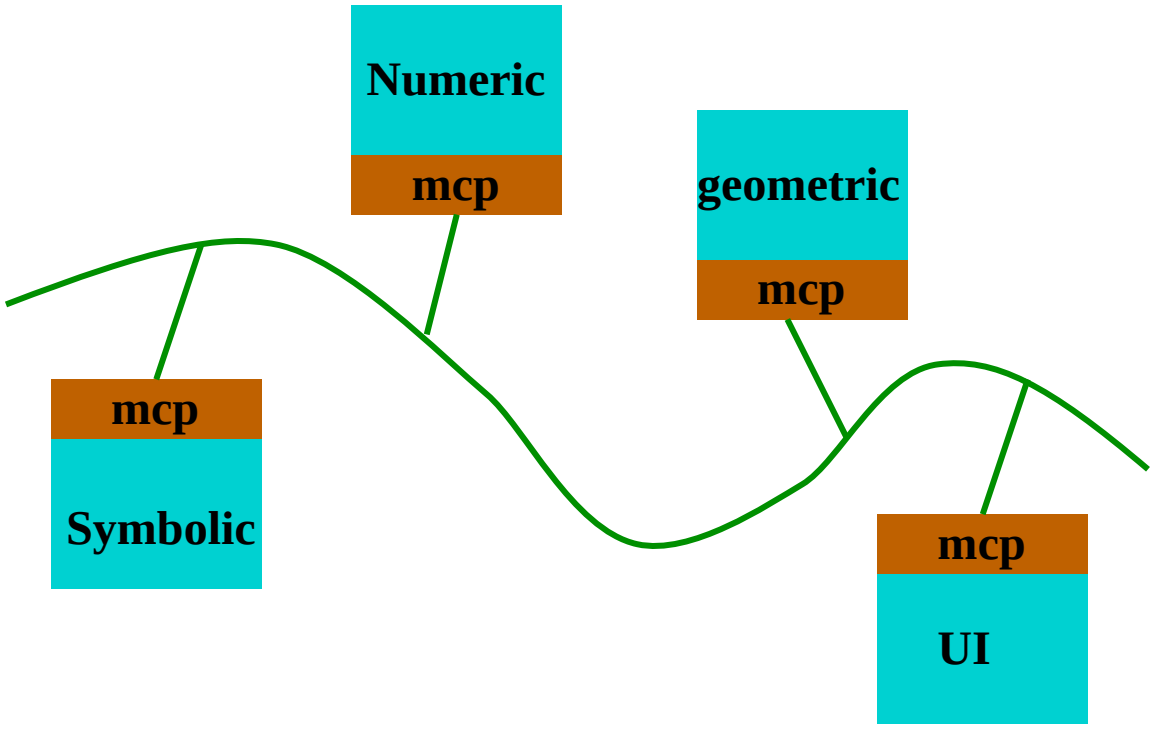

Figure 6: MCP Use in PSE

6.2 MCP

The *Mathematical Computation Protocol* is modeled after HTTP. Each MCP message consists of a header and a body. Each header entry is a key-value pair. The header and body are separated by an empty line. The first line of an MCP message is in the form

`Client MCP` *version*
`Server MCP` *version*

which identifies the sender being a client or a server and the MCP protocol version number.

Supported header keys include

- `Status:` normal, error, ready, busy, terminating
- `ControlRequest:` *type*, optional
- `ControlResponse:` *type*, optional
- `Sequence:` linear sequence number (control and compute sequenced separately)
- `Content-Type:` body type (e.g. application/x-mp, or text/MathML) default text/plain
- `Content-length: bytes`

- `Transfer-Encoding`: if any, default none (useful for email transfer)

Here is an example client computation request

```
Client MCP 1.0
Status: normal
Sequence: 1
Content-Type: application/x-mp
Content-length: 356

Body
```

A sample server response is

```
Server MCP 1.0
Status: normal
Sequence: 1
Content-Type: application/x-mp
Content-length: 4000

Body
```

A client control request may look like

```
Client MCP 1.0
Status: normal
ControlRequest: disconnect
Sequence: 7
```

To which the server may respond

```
Server MCP 1.0
Status: normal
ControlResponse: Bye
Sequence: 7
```

It is also possible for an IAMC server to present requests to the IAMC client, mostly to obtain information from the user. For example the server may need to know if a variable is real, positive, or if $x > y$. The server may send a *user selection* request, a *menu selection request*, etc., to the client. The server may also wish to request the setting of a *cookie*. A cookie is any information the server wishes the client to store and transmit back to the server next time it connects. This can be easily accommodated by

```
Server MCP 1.0
Status: normal
ControlRequest: SetCookie
```

```
Sequence: 5
Content-type: application/binary

Body (the cookie)
```

A client can present a saved cookie with

```
Client MCP 1.0
Status: normal
ControlRequest: Cookie
Sequence: 1

Body (saved cookie)
```

7 Flexible Technical Content

The MCP protocol uses the `Content-Type:` header key to indicate the type of the body of the message. Mathematical expressions can be coded in MathML which is a recommendation from the W3 consortium, or MP which is an efficient binary encoding format developed at ICM/Kent. The scheme obviously accommodates any other formats or data needed to provide the service at hand, for example, sending GIF files for displaying curves and surfaces of mathematical functions.

Here we will briefly introduce MathML and MP, the formats that are likely to be the main workhorse for IAMC.

7.1 MathML

MathML supports presentation markup and content markup of mathematical expressions. The former is for screen rendering and the latter for performing computations. For example, the expression

$$x^2 + 4x + 4 = 0$$

is coded in presentation markup as

```
<mrow>
  <msup> <mi>x</mi> <mn>2</mn> </msup>
  <mo>+</mo>
  <mn>4</mn> <mi>x</mi>
  <mo>+</mo>
  <mn>4</mn>
  <mo>=</mo>
  <mn>0</mn>
</mrow>
```

And the presentation markup of

$$\sqrt[3]{1 \; - \; \frac{x}{2}}$$

is

```
<mroot>
  <mrow>
    <mn>1</mn>
    <mo>-</mo>
    <mfrac> <mi>x</mi> <mn>2</mn> </mfrac>
  </mrow>
  <mn>3</mn>
</mroot>
```

MathML content markup employs a *prefix notation* to capture the *computational content* of mathematical expressions. The general form is

`<apply>` *Operator operand1 ...* `</apply>`

where an operator is something like `<log\>` or `<cos\>`. For example:

$$\sin(x) \; + \; 9$$

has the following MathML content markup code

```
<apply> <plus/>
          <apply> <sin/> <ci> x </ci> </apply>
          <cn> 9 </cn>
</apply>
```

And the second derivative of $f(x)$

$$\frac{d^2}{dx^2} \; f(x)$$

is content coded as

```
<apply><diff/>
  <apply><fn> f </fn>
    <ci> x </ci>
  </apply>
  <bvar>   <ci> x <ci> </bvar>
  <degree> <cn> 2 </cn> </degree>
</apply>
```

7.2 MP Format

The Multi Protocol (MP) [6], [7] is a result of collaboration among S. Gray, N. Kajler and P. Wang. MP is a format for efficient communication of mathematical data among scientific computing systems. The MP design strives for efficiency, flexibility, and extensibility in scientific data exchange. Here are some highlights.

- Binary parse tree data encoding – MP defines a set of basic types: text-based types (identifier, operator, string), binary types (single and arbitrary precision integer and real), a raw type for transmitting uninterpreted data, plus the meta and MP-operator types for protocol management purposes. More complicated data such as mathematical expressions, data structures, function calls, procedures, etc. are represented by *MP annotated trees* (`MaT`s). A `MaT` is basically a binary-encoded parse tree whose nodes may have attached *annotations*. `MaT` is simple yet powerful enough for most, if not all, practical needs.

- Annotations – Each `MaT` node may be *annotated* with supplementary information. Annotations may provide information about how a data item is to be interpreted (e.g., units of measurement - miles, centimeters, Joules), the original source of the data, and any application specific attributes. MP-defined annotations are understood by any system using MP. Application-defined annotations can be established to fill problem-specific needs.

- Optimizations – In addition to efficient binary encoding of basic data types, MP uses several techniques to reduce the size of encoded data including *common subexpression elimination* and *repeated data patterns* using the *MP prototype*.

- Dictionaries – The `MaT` deals with mathematical expression syntax. MP uses a separate *dictionary* mechanism to handle semantics. A dictionary is a human-readable document containing a list of items together with their meaning. A definition may be formal or informal, but must be sufficiently precise. Basically, a dictionary defines a name space within which names have preassigned meanings. Dictionaries are identified with names and can be organized into hierarchies. Here is a sample dictionary entry

```
17  Pi  Circumference / diameter of circle
```

For example, the function

$$(f := x \rightarrow x**3 - 1)_{\text{(source maple)}}$$

is encoded in MP as follows

```
op      0  :=      2
source
string  0  maple
```

```
id      0  f
op      0  ->      2
id      0  x
op      0  -       2
op      0  **      2
id      0  x
int     0  3
int     0  1
```

where strings are used instead of numbers to make the example human readable.

8 Implementation

The design of IAMC is not final. But one way to test and refine the design is to proceed with the implementation of key parts of the IAMC system to gain feedback and experience. Mr. Wei Wu is finishing a master thesis that focuses mainly on building a proof-of-concept system for IAMC. A simple IAMC client-server pair is written in Java, using MP for mathematical data encoding and transfer. The server provides Internet access to the MAXIMA symbolic computation system at Kent.

The client can parse infix mathematical input, convert input to internal (prefix) notation, translate that to MP, and send a computation request to the server. Results are displayed in infix, prefix, or LaTeX and can be saved to files for later use.

The server can be launched by the `inetd`. It deals with the client on the one hand and the external MAXIMA engine on the other. For this experiment, the server and client simply send and receive MP encoded messages. The server converts MP data to infix and feeds that to MAXIMA as input and translates output from MAXIMA to MP before sending back to the client. A single LISP [11] function of MAXIMA is modified to make it output results in LISP prefix form which makes server conversion to MP rather trivial.

On-going work on on IAMC iclude design refinements, a GUI for the client, and a reference implementation for the MCP protocol.

References

[1] J. Abbott, A. Diaz, and R. S. Sutor, "A Report on OpenMath," ACM SIGSAM Bulletin, pp. 21-24, March 1996.

[2] N. Borenstein and N. Freed, *MIME: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, the network working group RFC-1521, Sept., 1993.

[3] S. Dalmas, M. Gaëtano, and S. Watt, "An OpenMath 1.0 Implementation," Proceedings, ISSAC'97, ACM Press, pp. 241-248, 1997.

[4] R. J. Fateman, "Network Servers for Symbolic Mathematics," Proceedings, ISSAC'97, ACM Press, pp. 249-256, 1997.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee *Hypertext Transfer Protocol – HTTP/1.1*, RFC-2068, January 1997.

[6] S. Gray, N. Kajler and P. Wang, "MP: A Protocol for Efficient Exchange of Mathematical Expressions," Proceedings, ISSAC'94, ACM Press, pp. 330-335, 1994.

[7] S. Gray, N. Kajler and P. S. Wang,

[8] "Design and Implementation of MP, a Protocol for Efficient Exchange of Mathematical Expressions" *Journal of Symbolic Computation*, Vol. 25, Issue 2, Academic Press, Feb. 1998, pp 213-238

[9] P. Ion and R. Miner, ed., *Mathematical Markup Language*, W3C Working Draft, Jan. 6 1998.

[10] D. Raggett, J. Lam, I. Alexander, and M. Kmiec, *Raggett on HTML 4*, Addison-Wesley, 2nd edition, ISBN 0-201-17805-2, 1997.

[11] T. Yuasa and M. Hagiya, "Kyoto Common Lisp Dictionary", Kyoto University, KCL on-line documentation, 1986.