Design and Protocol for Internet Accessible Mathematical Computation

Paul S. Wang*
pwang@mcs.kent.edu
http://icm.mcs.kent.edu/∼pwang
ICM/Kent State University

January 22, 1999

Abstract

Mathematical computing can become easily accessible and conveniently usable on the Internet. The distributed Internet Accessible Mathematical Computation (IAMC) system can supply mathematical computing powers widely through TCP/IP, the Web, or email. The overall IAMC architecture, a feasibility study, elements of the *Mathematical Computation Protocol* (MCP), the design of IAMC client and IAMC server, the server interface to compute engine, and the API design for a Java implementation of MCP are presented.

1 Introduction and Background

The Internet and the World-Wide Web make many kinds of information and services easily accessible. Ad-hoc methods have been used to make mathematical computing available on the Internet. At the University of California, Berkeley [7], for example, an extensive table of integrals is network-accessible. At Kent State University, the `SymbolicNet` Web site[1] offers live system demos accessible either through simple CGI programming or by accessing remote X servers. For providing mathematical computing on the Internet, these approaches are limited in generality, hard to deploy widely, and short on interoperability.

The importance of making technical/mathematical communication available 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[2] [12] defines an SGML language for markup of mathematical expressions. Both presentation (display layout) and content (computation semantics) markup are supported (Section 7.2).

*Work reported herein has been supported in part by the National Science Foundation under Grant CCR-9721343

[1] `http://SymbolicNet.mcs.kent.edu`

[2] `http://www.w3.org/TR/WD-math-980106/`

NetSolve, a joint project between the University of Tennessee and the Oak Ridge National Laboratory, makes scientific computation packages, mostly numerical in nature, available to users through a variety of interfaces and to Web users through a Java Applet. NetSolve *agents* register computation *resources* and refer clients to them based on capabilities of the resources, computational efficiency and fault tolerance considerations. The Applet client being tested is available on the NetSolve Web site[3].

The IBM digital publishing group has released the experimental *Techexplorer* [6], 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. Techexplorer also allows a user to send expressions to a fixed compute server for evaluation. The WebEQ from Geometry Technologies Inc.[4] can display WebTeX and a set of MathML by converting them to LATEX images. The W3 Consortium's Amaya Web browser[5] demonstrates a prototype implementation of MathML which allows users to browse and edit Web pages containing mathematical expressions. Together with the rest of the Web page, these expressions are manipulated through a WYSIWYG interface.

In addition to *mathematical content viewing* on a Web page, users need to have easy access to *mathematical computing* on the Internet. The mathematical results obtained from one computation service ought to be usable by another. Accessing a math-oriented computing service should be as simple as entering a command, retrieving a Web page, or sending email.

The Institute for Computational Mathematics at Kent State University, together with collaborators at other institutions, is pursuing the IAMC research project to

- Make math-oriented data and services easily and widely accessible on the Internet – directly, via the Web, and by email
- Support interactive use of user-designated remote *compute servers* almost as if they were local programs
- Provide effective and efficient communication of mathematical data over the Internet
- Allow exchange and further processing of computational results among different compute servers

To achieve these goals, we investigate the architecture, protocol, and implementation of a distributed IAMC system. For IAMC, major efforts include defining a *mathematical computation protocol* (MCP), implementing a Java class library for MCP, and building prototypes for a typical IAMC client and a customizable IAMC server.

It is hoped that IAMC, when developed and deployed, can make research, experimental, parallel/super computing, one-of-a-kind, demonstration, or commercial software systems that deal with mathematics in any technical discipline easily accessible on a global basis.

The IAMC effort is on-going. Here, the overall architecture, designs for the IAMC client, the IAMC server, and the MCP that connects them will be presented. Implementation experiments on

[3] `//www.cs.utk.edu/~casanova/NetSolve/`
[4] `http://www.webeq.com/webeq/`
[5] `http://www.w3.org/Amaya/`

a simple stream socket based system are described. Design of a Java API for MCP is also included. Results reported here revise, refine, and expand earlier efforts on IAMC presented at ASCM'98 [13] as an invited talk.

2 The IAMC Approach

Basically, each *IAMC server* (Isv) provides a specific computational service and has a URL in the form

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

where *port* specifies the port number of the per-host *IAMC daemon* which will invoke the Isv specified by the *server-id* (Fig. 1). An Isv can perform the computational tasks either directly or through a separate compute engine. One Isv may transparently use other IAMC servers to perform subcomputations.

An end user accesses IAMC through an *IAMC client* (Icl). An Icl contacts an Isv and supports interactive access to a set of computations. The same Icl may interact with multiple IAMC servers concurrently. When supported, a user may also ask the Icl to request a computation via email. Each email requests a sequence of computations and obtains one final result by return email.

Client-server communication uses the *Mathematical Computation Protocol* (MCP) designed specifically for the purpose. An MCP message allows flexible content types and IAMC clients and

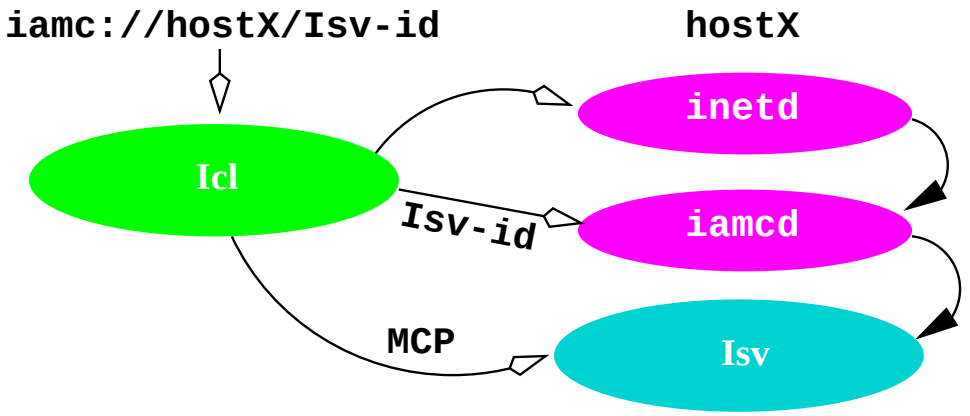

Figure 1: Connecting to An IAMC Server

servers will be able to exchange mathematical data through a common math-encoding.

IAMC services can be available in different ways. A user can invoke an Icl and connect directly to any Isv specified by a URL. IAMC service providers can make their services easily available in Web documents via hyperlinks. A Web browser can launch an Icl to connect to a target Isv automatically. This way, users can find compute servers through Web search engines. Applications of IAMC include research in mathematics, science, and engineering; mathematical education and distance learning; remote scientific databases; making parallel/super computing more accessible; computing via NetPC for high school or occasional users.

3 Architecture Overview

The IAMC system consists of the following components:

- Icl — The end-user agent for accessing IAMC services.
- IAMC daemon (`iamcd`) — The Isv launch agent. It uses a stream socket at a prescribed port and starts any local Isv requested . The `iamcd` can listen to incoming connections or be managed by the `inetd`.
- Isv — The IAMC server providing a specific computational service. An Isv can be launched by the `iamcd`, a CGI program, or an autonomous mail processor such as `procmail`.
- MCP — The *Mathematical Computation Protocol* for linking IAMC clients and servers. An MCP library can be used by the Icl and the Isv alike.
- Mathematical Data Encoding — Common Mathematical data encoding formats. MP and MathML are the common *math-encoding formats* for exchanging mathematical data and formulas within IAMC. Since the MCP protocol also allows any other mathematical format, IAMC clients and IAMC servers can use additional formats.
- Compute engine — An independent computation agent. IAMC servers can interface to existing or newly developed systems to perform a wide range of technical, scientific, and mathematical computations.

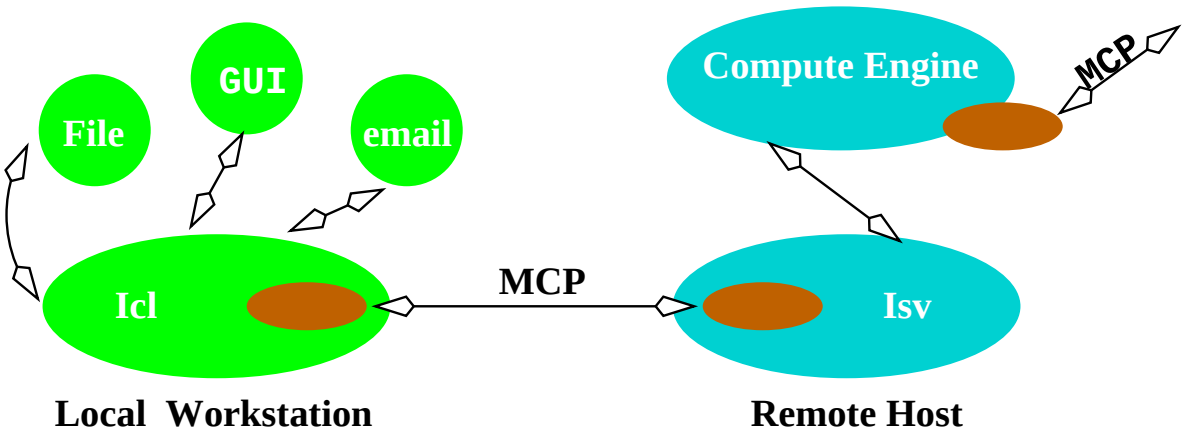

Figure 2: IAMC Architecture Overview

Figure 2 shows an overview of the IAMC architecture. Now let's consider the designs for a typical client and server.

4 IAMC Client Design

An Icl, as the end-user agent, interfaces to the user on the one hand and to the Isv on the other. A simple Icl can provide a no-frills command-reply interface. A sophisticated Icl may offer a GUI with many functionalities and conveniences. For example, an Icl may offer interactive display of mathematical curves and surfaces. The set of points may be supplied by an Isv, but axes, labeling, viewing angle, hidden line removal, zooming, etc. can be handled by the Icl.

A good user interface can make mathematical computations much easier to do. A survey of previous work in mathematical user interfaces can be found in [11]. Experiences gained on the SUI (System-independent User Interface) work [4, 5] are valuable for the Icl design.

Duties of a typical Icl include:

- Connects to specified remote/local servers
- Communicates with servers using MCP
- Receives commands and mathematical expressions as input from the user and/or from a file
- Checks and parses user input, converts such input to appropriate representations
- Edits input, saves input into files
- Encodes computation requests in math-encoding and sends them to an Isv
- Receives and displays results encoded in math-encoding from an Isv
- Controls and manages computational transactions and sessions
- Treats/converts results obtained to useful formats: MathML, MP, GIF, HTML, OpenMath, infix, prefix, LaTeX, and so on
- Requests and displays server-supplied help/documentation information to the user
- Keeps track of user input and history
- Stores/retrieves mathematical data files from local file system
- Provides an interface for external plug-ins allowing an open-ended extension to the types of data that can be handled

An Icl may also provide an email interface for IAMC. Figure 3 shows the object-oriented design of a prototype IAMC client. Top-level Java classes are shown.

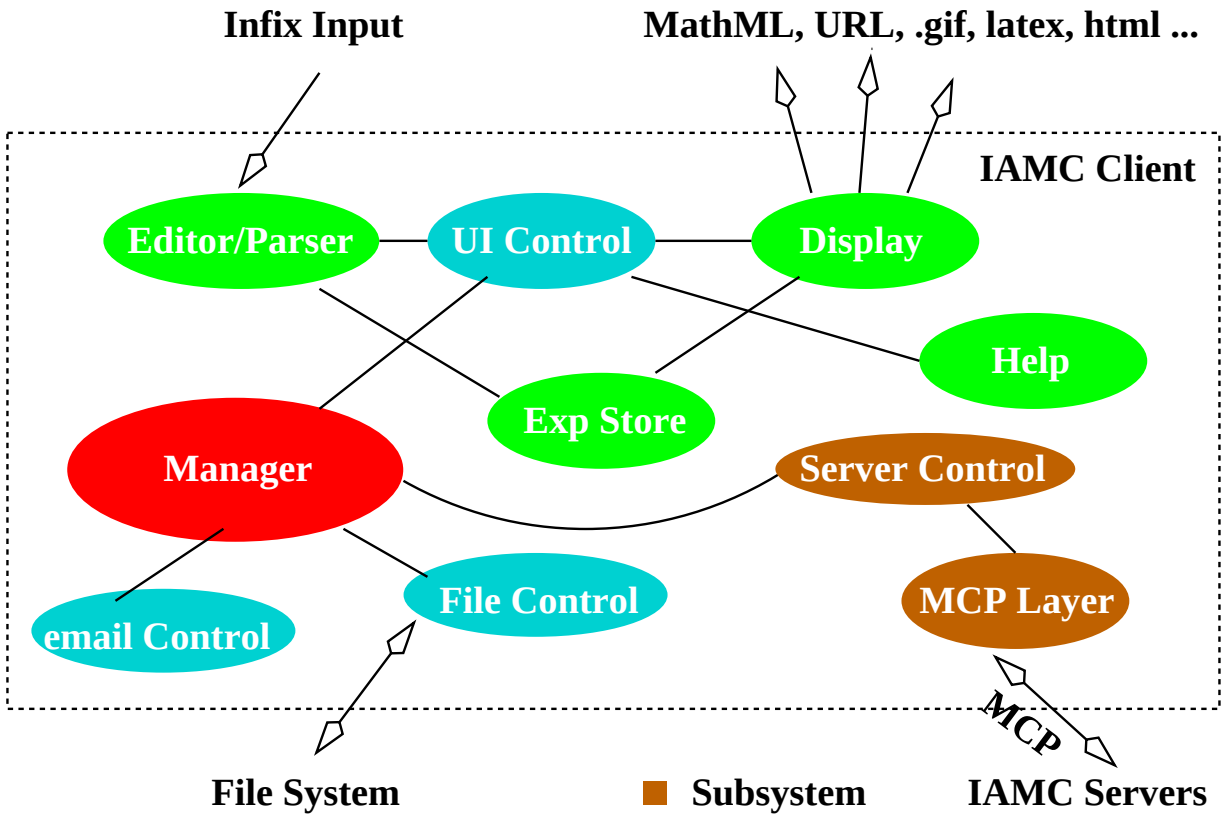

Figure 3: IAMC Client

4.1 Command Templates

To make mathematical operations easier for the end user, an Icl can offer *command templates* through menus and/or buttons. A template makes issuing mathematical commands simple. For instance, an integration command template may display, together with textual explanations,

$$\int_a^b \; f(x)\; dx$$

followed by a dialogue box for entering an integration command (Fig. 4).

To make usage even easier, frequently used mathematical operations such as integration, differentiation, summation, substitution, solution of equation(s), curve and surface plotting, etc. can follow a uniform syntax across all IAMC clients. For example, the syntax and semantics defined by MP (section 7.2) or OpenMath [3] can be adopted. Other commands can use *Isv-supplied templates* (Section 6.5).

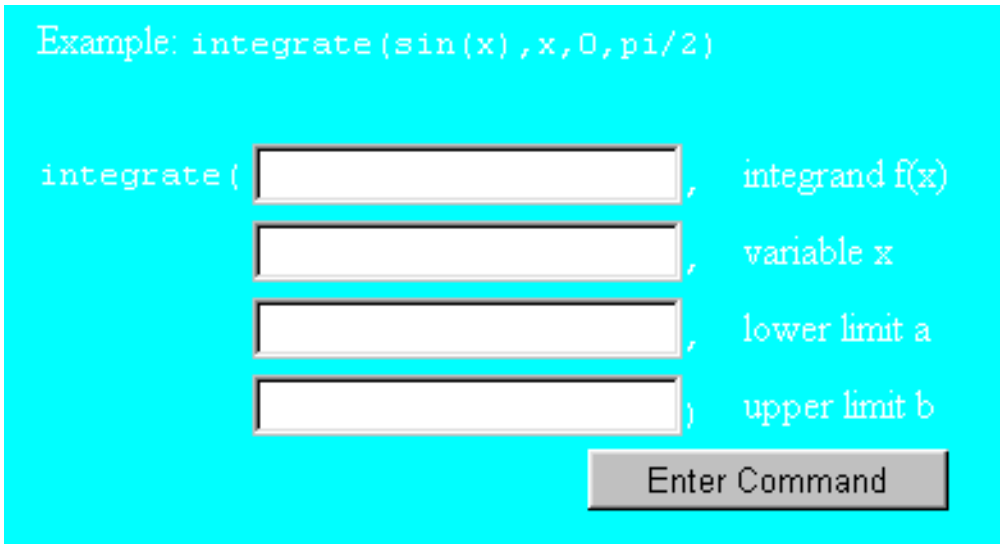

Figure 4: Integration Command Template

5 IAMC Server Design

An Isv makes a set of computations available and may be spawned by an `iamcd`, a CGI program, or an email processor. Once connected, the Isv-Icl pair cooperates to supply computation power across the Internet. The client-server communication and the computation control parts of the Isv should be reusable. Computation-specific functionalities are performed either directly in the server or indirectly through an external compute engine. A typical Isv will be responsible for these tasks.

- Receives requests sent by the Icl.
- Communicates with Icl via MCP.
- Decodes math-encoded Icl requests.
- Performs requested operations and computations.
- Sends responses back to the Icl.
- Controls and interfaces to external compute engine, if so configured.
- Provides end-user help and usage documentation to the Icl.
- Logs operations, errors, and statistics.

Figure 5 shows the object-oriented design of an Isv.

Many general- and special-purpose mathematical compute engines exist. New engines will continue to be built. An Isv can be used to adapt a stand-alone engine under IAMC and to make

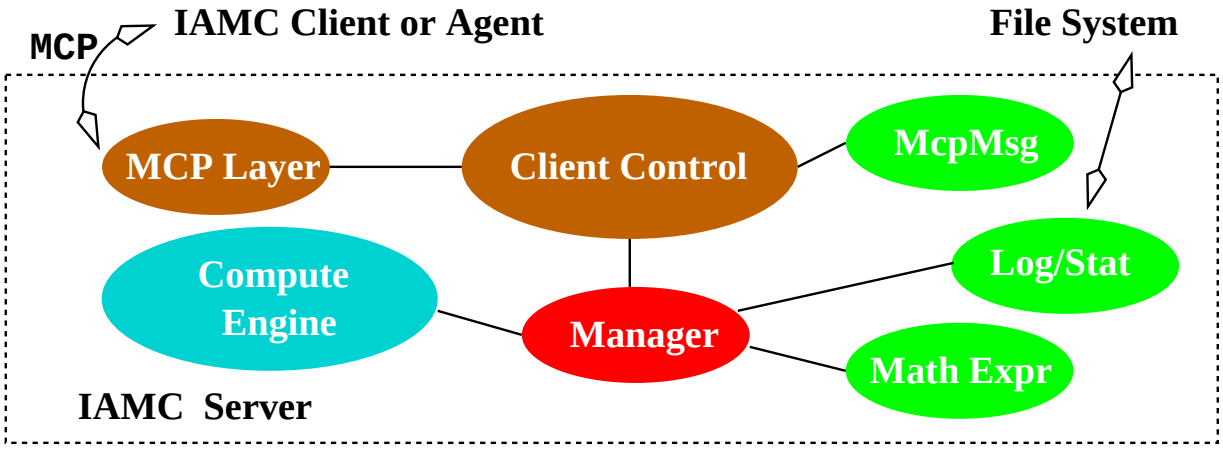

Figure 5: IAMC Server

the computational power widely available on the Internet. Through IAMC, results from one engine become easily usable on another.

Setting up an Isv and adapting a stand-alone system should be made easy. To achieve this goal, we can design and implement a reusable Isv, say in Java, with a flexible *external engine interface* (EEI) which can be customized to adapt existing engines. The EEI for a reusable Isv will be considered next.

5.1 Server Interface to External Engine

We shall assume the Isv and the compute engine run as separate processes on the same host computer and they perform interprocess communication locally. Basically, the Isv receives computation requests from an Icl and sends the request in a suitable form to the external engine for processing (Fig. 6). Incoming expressions may be encoded in one of two known formats (MP and MathML). The term *math-encoding* in this paper can refer to either of these formats.

A computation request received by an Isv can be in one of two forms:

- *literal command* — a character string that will be sent, directly without change, to the engine's input interface.
- *math-encoded command* — a command in math-encoding which can be sent either directly to the external engine, if it accepts the format, or after being converted to an acceptable form.

A class for converting math-encoding to infix notation would be part of the EEI. The conversion class can be customized through class extension.

To work with the EEI, an external engine can receive:

- Command strings that the engine normally admits as a stand-alone program
- Mathematical expressions in infix notation

- Optionally, math-encoded expressions for efficiency

Having obtained a mathematical result from the external engine, an Isv must math-encode the result before sending it to the client. This conversion is engine dependent. To make writing the custom conversion program easy, the EEI can supply a math-encoding class that uses a customizable decoding object which can be made to understand the mathematical output format of a target engine. Results 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 an image in GIF, or generated Fortran code in plain text.

In addition to the command/result interface, the EEI also has a help/documentation interface to the external engine. A user can obtain command syntax and usage examples for a target compute engine through the help facility provided by an Icl. The engine may return the requested help or documentation information in HTML, plain text, PDF, or a URL. The Isv passes these back to the Icl directly. An external engine also supports a `commandTemplate(`*commandName*`)` command which returns the usage template for the specified *commandName*.

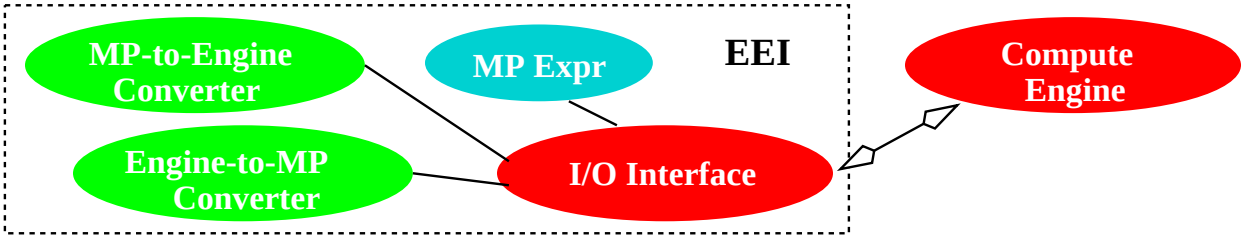

Figure 6: Server Interface to Compute Engine

The EEI must also handle queries from the external engine to obtain specific information from the end-user and interrupts from the Isv to the engine to abort computations.

Having discussed the IAMC client and server, we now turn our attention to the Mathematical Computation Protocol they use.

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 aims to be an effective yet flexible protocol whose primary purpose is to support IAMC. But, it may find other uses. Here are some important considerations for MCP:

- Meeting client-to-server and server-to-client requirements
- Supporting various data-transfer and data compression encodings

- Allowing different mathematical representation formats
- Employing two-way, sequenced, reliable connection for computation sessions
- Assuming peer-to-peer interactions, no multiple client connections to the same server

Between the server and the client, we distinguish between three types of requests:

1. *Computation request*: a request to perform, affect, or obtain information about a mathematical computation. Such requests produces computation results, side effects, or help/documentation. Computation requests are generated by the Icl.
2. *Control request*: a request for controlling the operations or functionalities of the client or server. Such requests are for managing the Icl and the Isv.
3. *Dialog request*: a request directed to the end-user to obtain answers to specific questions or choices. Dialog requests are generated by the compute engine or the Isv.

For example, *compute the first derivative*, *send help information on integration*, and *generate 2D surface plot* are computation requests while *initialize*, *login*, *report server/client status*, and *disconnect* are control requests.

Now the client and server requirements can be listed.

- An Icl needs to send/receive control requests and responses; send computation commands; receive computation results in various encoding forms; issue commands synchronously (waiting for result before next command) and asynchronously (no waiting); receive results synchronously and asynchronously; abort an on-going computation; handle dialog requests.
- An Isv needs to send/receive control requests and responses; receive synchronous and asynchronous commands; send computed results in various formats (MathML, MP, OpenMath, GIF, PDF, ...); use different transfer encodings; allow various data compression methods; send back results; query the end-user for information.

Additional practical service requirements include authentication, accounting, and security.

6.2 MCP Message Format

The Icl-Isv pair exchanges MCP messages to communicate. The *Mathematical Computation Protocol* message format is modeled after HTTP [8]. Each MCP message consists of a *header* and a *body* (optional). 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 one of

`Command` — An Icl sends a `Command` to request a computation.

`Result` — An Isv sends a `Result` message in response to a `Command` transmitting any results of the computation.

`Dialog` — An Isv sends a `Dialog` request to solicit information from the end user. An Icl sends a `Dialog` response to return the information collected from the end user.

`Control` — The previous three are *computation messages*. Icl and Isv sends `Control` messages to conduct non-computation business for the control and management of the MCP session.

Control messages do not contain computation commands or results. Command or result messages may contain control headers.

After the first line, the header may have zero or more lines of key-value pairs. There are control, computation, and general headers. Some control headers are

- `ControlRequest:` *seq_no*`-`*cntl_string*
- `ControlResponse:` *seq_no*[`-`*cntl_string*]
- `Version:` MCP/*version_number*
- `Accept:` list of acceptable result types
- `CanDo:` list of supported commands

Headers such as `Version`, `Server-name`, `User-agent`, `Accept`, `CanDo` are used only in initialization messages. For example, an Icl can use the header

```
ControlRequest: C1-initialization
Version: MCP/1.0
User-agent: SuperSolver
```

with a sequence number `C1` (client 1) to request initialization after getting connected to an Isv. And the Isv may use

```
ControlRequest: S1-login
```

to request user login as part of the response to the `C1` request. Or an Isv may send

```
ControlRequest: S17-prompt
```

to prompt the client for the next command. An Icl may inquire about the Isv's readiness for another command with

```
ControlRequest: C23-readyStatus
```

to which the reply may be

```
ControlResponse: C23-busy
```

Note client and server control requests are sequenced independently.

Computation messages are used to send commands and results. Computation headers include

- `Command-string: "`*any-string*`"`
- `Mode: sync` (or `async`, default is `sync`)
- `Send-result: yes` (or `no`, default is `yes`)
- `Sequence:` sequence number
- `DialogRequest:` *format*
- `DialogResponse:` *format*

If a `Command-string` is given then it is the literal command to be used or sent to an external compute engine. Otherwise, the command is given as the body of the message in a specified content type. The Isv may need to convert this type to an appropriate form before using it as a computation command. An Isv issues command in synchronous mode if it waits for the result from the Isv before sending another command and in asynchronous mode if it continues without first receiving the result. The sequence number identifies which result corresponds to which command.

Dialog request and response headers are used by the Isv to query the end user and obtain information. For this purpose, a dialog request can use the HTML form format and the dialog response can use the well-known form data format (`x-www-form-urlencoded`).

Last but not least, general headers include:

- `Status: normal`, or `error` *err-number err-string*
- `Content-type:` *body type*
- `Content-length:` *number of bytes*
- `Transfer-encoding:` *encoding*

A response message will always include the `Status` header.

6.3 Sample MCP Communication Scenarios

Here is a sample communication sequence between an Icl and an Isv. Messages from the Icl are indented. Long lines have been broken to fit the page.

1. Initialization after getting connected:

```
Control
ControlRequest: C1-initialization
Version: MCP/1.0
User-agent: MathBrowser
Accept: application/x-math-MP,text/MathML,
image/GIF,text/HTML,application/PDF
```

```
Control
ControlResponse: C1
Version: MCP/1.0
Status: normal
Server-name: PolyFactor
Greeting: Performs univariate and multivariate polynomial
          factoring over the integers
CanDo: factor,expand,ratsimp
[ ControlRequest: S1-login ]
```

2. First computation command from Icl (assign value to p)

```
        Command
        Sequence: 1
        Command-string: "p : 4*x^2-1"
        Send-result: no

Result
Sequence: 1
Status: normal
```

3. Second computation command from Icl (get factors of p)

```
        Command
        Sequence: 2
        Command-string: "factor(p)"

Result
Sequence: 2
Status: normal
Content-type: application/x-math-MP
Content-length: 26

<body contains (2*x + 1)*(2*x - 1) in MP format>
```

4. Icl terminating

```
        Control
        ControlRequest: C2-Disconnect

Control
ControlResponse: C2-Disconnect
Status: normal
```

6.4 Querying the End User

To obtain specific information from the end user, an Isv sends a `Dialog` request to the Icl. For instance the server may need to know if a parameter is positive negative or zero. Here is an example:

```
Dialog
DialogRequest: HTML-form
Content-type: text/HTML
Content-length: 145

<HTML form for user>

        Dialog
        DialogResponse: form-data
        Content-type: application/x-www-form-urlencoded
        Content-length: 45

        <name=value pairs separated by &>
```

6.5 Help and Command Template

On behalf of the end user, an Icl may also request help/documentation information from an Isv:

```
        Command
        Sequence: 14
        Command-string: "help(integration)"

Result
Sequence: 14
Status: normal
Content-type: text/plain
Content-length:  245

integrate(f(x),x, a, b) computes the exact definite
integral of f(x) from a to b; integrate(f(x),x) computes
the exact antiderivative of f(x); romberg(f(x), x, a, b, eps)
computes the numerical quadrature of f(x) from a to b with
accuracy eps.
```

More complicated help may involve giving the user several choices on topics through a dialog request, or sending back an HTML document with links to further help information.

The Icl can also request a command template from the Isv with

```
        Command
        Sequence: 25
        Command-string: "commandTemplate(limit)"
```

and get a reply from the Isv as follows

```
Result
Sequence: 25
Status: normal
Content-type: application/x-mcp-CommandTemplate
Content-length: ...

command: limit(f,x,pt)
effect: returns the limit of f as x approaches point pt
example: limit(sin(x)/x,x,0)
example: limit((1+1/x)^x,x,inf)
arg: f-an expression involving the variable x
arg: x-an identifier representing a variable
arg: pt-a constant expression not involving x,
     or INF (positive infinity),
        MINF (minus infinity),
        INFINITY (infinity)
```

The command template is specified in a straightforward manner in the message body as shown.

7 Flexible Technical Content Types

The MCP protocol uses the `Content-type:` header to indicate the type of the body of the message, allowing all types of message content. Mathematical expressions can be coded in MathML, a HTML-style representation developed by the MathML group of the W3 consortium. Or they can be in MP, an efficient binary encoding format developed at ICM/Kent. MathML is convenient for small mathematical expressions. MP is good for larger expressions. IAMC clients and servers will support at least MathML and MP for normal operations. OpenMath [1, 3] can be an alternative to MathML or MP.

Obviously, other data formats are useful as well. For example, GIF can be used for displaying curves and surfaces of mathematical functions and plain text or HTML can be used for help/documentation purposes.

Here we will briefly introduce MathML and MP.

7.1 MathML

MathML uses textual *markup tags* to support `presentation markup` and *content markup* of mathematical expressions. MathML can be used directly in HTML documents (within `<math>`

elements). 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: $\cos(x) + 7$ has the following MathML content markup code

```
<apply> <plus/>
          <apply> <cos/> <ci> x </ci> </apply>
          <cn> 7 </cn>
</apply>
```

And the second derivative $\frac{d^2}{dx^2} f(x)$ 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) [9, 10] is a result of collaboration among S. Gray, N. Kajler and P. Wang. MP facilitates efficient communication of mathematical data among scientific computing systems. MP defines a binary mathematical expression encoding format (the MP format) and provides data communications mechanisms as well.

Some characteristics of the MP format are:

- 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 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 for π:

```
<ConstDef>
  <DefName> Pi </DefName>
  <DefTag>  3  </DefTag>
  <Description> Circumference/diameter of circle. </Description>
  <CMP> 3.1415926535897932385, approximation to 20 digits </CMP>
</ConstDef>
```

For example, the function

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

is encoded in MP as follows

```
op      1  :=      2      (1 annotation 2 operands)
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 integer indices to make the example human readable.

8 Implementation

8.1 Proof-of-concept System

The design of IAMC is evolving. 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. In his master thesis entitled *Experiments with Internet Accessible Mathematical Computation* [14], Wei Wu describes the building of just such a proof-of-concept system for IAMC. A simple Icl-Isv pair is written in Java, using MP for mathematical data encoding. The implementation provides Internet access to MAXIMA, a Common Lisp version of the symbolic computation system Macsyma.

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 `.mp` files for later use.

The server can be launched by the `inetd`. It communicates 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 data via stream sockets without using the MCP protocol. The server converts MP data to infix and feeds that to MAXIMA as input and translates output from MAXIMA to MP before sending it back to the client. A single LISP function of MAXIMA is modified to output results in LISP prefix form which makes server conversion to MP rather trivial. In this work, MP-1.1.3 has been used.

8.2 Java API for MCP

The Mathematical Computation Protocol (MCP) will be implemented as a Java package that can be loaded by either an Icl or an Isv. The MCP layer in a server interacts with the MCP layer in a client to perform communication.

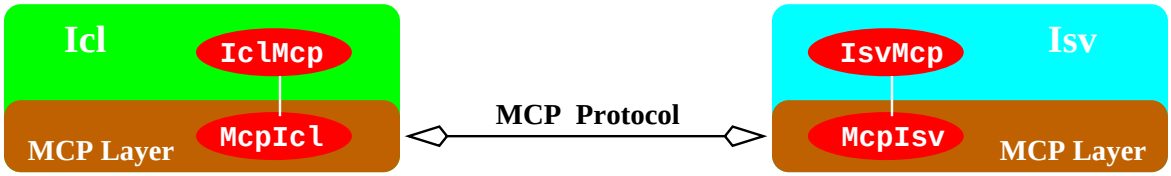

Figure 7: MCP Layer Interface

Major classes in the Java implementation of MCP include `McpMsg` (the MCP message), `McpIsv` (the MCP-to-Isv interface), `IsvMcp` (the abstract Isv-to-MCP interface), `McpIcl` (the MCP-to-Icl interface), and `IclMcp` (the abstract Icl-to-MCP interface).

MCP-Isv Interface

In an IAMC server, the MCP-to-Isv interface is implemented by an `McpIsv` object and the Isv-to-MCP interface is supported by an `IsvMcp` object. The *peer objects* run as cooperating threads with the `McpIsv` thread at a higher priority. Each object invokes the other to pass along information such as a command or a result (Fig. 8.2).

The `McpIsv` class supports these methods:

- `public McpIsv(IsvMcp svr)`
  Constructor. Initializes an `McpIsv` object to receive data from the Icl via standard input and send data to the Icl via standard output. The argument `svr` is the peer object.
- `public McpIsv(InputStream in, OutputStream out, IsvMcp svr)`
  Constructor. Initializes an `McpIsv` object to perform I/O with the Icl via the given streams. The argument `svr` is the peer object.
- `public boolean putResult(McpMsg m)`
  Sends the computational result packed in the message `m` to the Icl. Returns `false` when failed.
- `public void ready(Boolean flag)`
  Indicates to the Icl that the server is or is not ready for additional work.
- `public McpMsg dialog(McpMsg m)`
  Sends the dialog request `m` for end user to the Icl and returns the information obtained.
- `public void terminate()`
  Indicates to the Icl that server is finished and disconnecting.
- `public boolean pingclient()`
  Requests client status. Returns `true` if client is ready and `false` if client is busy. Assumes client is dead after a preset timeout interval.

And the class `IsvMcp` supports the following methods for its peer:

- `public void command(McpMsg m)`
  Performs the computation command `m` which can be either synchronous or asynchronous. The result produced will be returned via a call to `putResult(McpMsg r)` later.
- `public void perform(McpMsg m)`
  Performs the computation command `m` without returning any result.
- `public void abort(int n)`
  Aborts command with sequence number `n`. This call requests the indicated command be stopped and its results be discarded.
- `public void quit()`
  Terminates computation session.
- `public void reset()`
  Resets the Isv to its initial state aborting all on-going computations.

MCP-Icl Interface

In an IAMC client, the MCP-to-Icl interface is implemented by an `McpIcl` object and the Icl-to-MCP interface is supported by an `IclMcp` object. The peer objects run in cooperating threads with the `McpIcl` thread at a higher priority. Each object invokes the other to pass along information such as a command or a result.

The `McpIcl` class supports these methods:

- `public McpIcl(String url, IclMcp cl)`
  Constructor. Initializes an `McpIcl` object to connect to the Isv given by the `url` and to cooperate with the specified peer object `cl`.
- `public McpMsg syncCommand(McpMsg cmd)`
  Sends the command `cmd` to the Isv in synchronous mode. Returns the computational result received.
- `public boolean asyncCommand(McpMsg cmd)`
  Sends the command `cmd` to the server in asynchronous mode. Returns `false` when failed. The result produced by this method will be received via a call to the `result` method of the peer `IclMcp` object.
- `void abort(int n)`
  Sends a control message to the sever to abort the prior computation request with sequence number `n`.

- `void terminate()`
  Disconnects from server.

- `public boolean pingserver()`
  Requests server status. Returns `true` if sever is ready and `false` if server is busy. Assumes server is dead after a preset timeout interval.

The class `IclMcp` supports these methods for its peer:

- `McpMsg queryUser(McpMsg q)`
  Processes the given query (dialog) to the user and returns data obtained from the end-user.

- `boolean putAsyncResult(McpMsg r)`
  Delivers the computational result packed in the message `r`. This method is called by the `McpIcl` object to deliver a result for an earlier asynchronous command.

9 Further Work

It is hoped that ideas reported here will generate discussions, feedback, and increased interest. A design welcome by many is critical if IAMC is to gain wide-spread use.

Continuing work on IAMC includes design refinements, building or adapting a GUI for the client, full specification of MCP, a Java class library for MCP, redesign and re-implement MP in Java, building MathML/MP and other format converters, and establishing various demonstration IAMC services.

The growing list of people involved/interested in IAMC include researchers from Kent State University, Ashland University, University of Cincinnati (USA), Ecole des Mines de Paris, Institute for Computational Mathematics in Paris (France), Instituto de Matematica of UFRGS (Brazil), Mathematics Mechanization Research Center in Beijing, and Lanzhou University (China).

10 Acknowledgements

Deep thanks go to Simon Gray (Ashland U.), Norbert Kajler (Ecole des Mines de Paris), and Dieter Schmidt (U. Cincinnati) whose ideas, discussions, suggestions, and comments contributed much to this paper.

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] Y. Doleh, *The Design and Implementation of a System Independent User Interface for an Integrated Scientific Computing Environment*, Ph.D. dissertation, Dept. Mathematics and Computer Science, Kent State University, May 1995.

[5] Y. Doleh and P. S. Wang, "SUI: A System Independent User Interface for an Integrated Scientific Computing Environment," Proceedings of the ISSAC'90, Addison-Wesley (ISBN 0-201-54892-5), Aug. 1990, pp. 88-95.

[6] S. S. Dooley, *Coordinating Mathematical Content and Presentation Markup in Interactive Mathematical Documents*, Proceedings, International Symposium on Symbolic and Algebraic Computation (ISSAC'98), Universität Rostock, Germany, 13–15 Aug. 1998.

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

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

[9] 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.

[10] S. Gray, N. Kajler and P. S. Wang, "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

[11] N. Kajler and N. Soiffer, "A Survey of User Interfaces for Computer Algebra Systems," *Journal of Symbolic Computation*, vol. 25, number 2, February 1998, pp. 127-160

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

[13] P. S. Wang, "Internet Accessible Mathematical Computation," Proceedings, 3rd Asian Symposium on Computer Mathematics (ASCM'98), Lanzhou University, Lanzhou P. R. China, August 6, 1998, pp. 1-13.

[14] Wei (David) Wu, *Experiments with I nternet Accessible Mathematical Computation*, Master's Thesis, Department of Mathematics and Computer Science, May 1998, ICM/Kent technical report ICM-199805-0003.