

# The MP Encoding for Distributed Mathematical Computations: An Object-oriented Design and Implementation

S. Gray

Department of Mathematics and Computer Science  
Ashland University  
Ashland, OH, U.S.A.

L. Tong and P. S. Wang\*

Department of Mathematics and Computer Science  
Kent State University  
Kent, OH, U.S.A.

**Abstract** *Mathematical computing can become easily accessible on the Internet. The distributed Internet Accessible Mathematical Computation (IAMC) system can supply mathematical computing power widely through TCP/IP, the Web, or email. A key enabling technology for distributed and parallel mathematical computation is a standard protocol for exchanging mathematical objects. The Multi Protocol (MP) allows cooperating applications to efficiently exchange mathematical data with a shared view of the data's syntax and semantics. While the MP design separates the encoding and transmission of data, the current C library for MP tied these closely. An object-oriented design of MP is described that decouples data representation from data transport to produce a pair of more reusable modules. The data representation module is also useful for the IAMC.*

*Keywords:* Mathematical Data Representation, OOP, Java, Internet

## 1 Introduction

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. On `SymbolicNet`<sup>1</sup> for example, one can find demonstrations of mathematical computations ranging from differentiation, integration, polynomial factorization, to plotting of curves and surfaces and `LATEX` code generation. Such access is made through direct TCP links, simple CGI programs, or remote X servers.

The importance of making technical and mathematical communication available on the Internet is underscored by the recent activities of the W3 Consortium and other groups to make publishing mathematical materials on the Web easy. The MathML working draft<sup>2</sup> defines an SGML language for markup of mathematical expressions. Both presentation (display layout) and content (computation semantics) markup are supported [?].

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

---

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

---

<sup>1</sup><http://SymbolicNet.mcs.kent.edu>

<sup>2</sup><http://www.w3.org/TR/WD-math-980106/>

client being tested is available on the NetSolve Web site<sup>3</sup>.

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  $\text{T}_{\text{E}}\text{X}/\text{\LaTeX}$ . 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.<sup>4</sup> is a Java applet that displays WebTeX and MathML in a Web browser. The W3 Consortium's Amaya Web browser<sup>5</sup> 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.

The Institute for Computational Mathematics<sup>6</sup> at Kent State University, together with collaborators at other institutions, is pursuing the IAMC (Internet Accessible Mathematics Computation) research project to address a number of critical issues for interactive access to mathematical computing on the Internet [?]. Section ?? gives a brief overview of IAMC.

One of the critical issues is establishing a common representation for mathematical expressions so scientific results can be transmitted and used among distributed systems with heterogeneous components. Various projects such as MathML [?], the Multi Protocol (MP) [?, ?] and OpenMath [?, ?], have been addressing this common representation issue.

MP is an efficient format using a binary encoding (Section ??). A C-coded library, MP-1.1.3, is available by public FTP (<ftp.mcs.kent.edu:/dist/MP>). Section ?? describes an object-oriented redesign and reimplementaion effort to make MP easier to use and better suited for integration with the emerging IAMC system. The object-oriented

design of the MP representation module is outlined in Section ?. The integration of MP in IAMC is presented in Section ?. Our plans to extend this work appear in Section ?.

## 2 IAMC Overview

IAMC is a distributed system to make mathematical computing easily accessible on the Internet [?]. It is designed to support direct, Web-based, or email access to remote mathematical compute engines. IAMC consists of the following components:

- Client — The end-user agent for accessing IAMC services.
- IAMC daemon (*iamcd*) — The Server launch agent. It uses a stream socket at a prescribed port and starts any local Server requested. The *iamcd* can listen to incoming connections or be managed by the *inetd*.
- Server — The IAMC server providing a specific computational service. A Server 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 Client and the Server alike.
- Mathematical Data Encoding — Common mathematical data encoding formats. Such common formats allow heterogeneous mathematical programs to interoperate through local-area and wide-area networks.

An overview of the IAMC architecture is given in Figure ?. Although IAMC allows different mathematical encodings through the **Content-type** mechanism, it promises to support *native* mathematical encoding formats. Application programs can rely on IAMC to provide basic operations for the native formats

---

<sup>3</sup><http://www.cs.utk.edu/~casanova/NetSolve/>

<sup>4</sup><http://www.webeq.com/webeq/>

<sup>5</sup><http://www.w3.org/Amaya/>

<sup>6</sup><http://icm.mcs.kent.edu/icm/index.html>

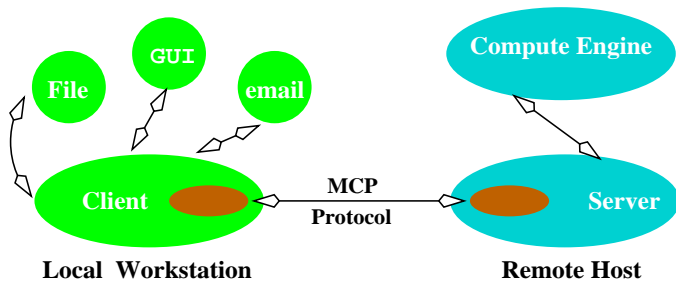


Figure 1: IAMC Architecture Overview

including encoding, format conversion, and display. IAMC aims to support MathML and MP as native mathematical formats.

### 3 The MP Encoding

MP is a result of collaboration among S. Gray, N. Kajler (Ecoles des Mines, Paris) and P. Wang. MP facilitates efficient communication of mathematical data among scientific computing systems. Some characteristics of the MP representation are:

- Binary parse tree data encoding – MP defines a set of basic types and mechanisms for constructing structured data. Numeric data (fixed and arbitrary precision floats and integers) are transmitted in a binary format (2's complement, IEEE float, etc.). Composite data (such as general expressions, polynomials, matrices, etc.) are represented as a linearized, annotated syntax tree (*MP Tree*) which is transmitted as a sequence of *node* and *annotation packets*, where each node packet transmits a node from the syntax tree.
- Annotations – The information carried in a node of an MP Tree may be augmented by an annotation. An annotation is either *supplementary* and can be safely ignored by the receiver, or may be *required* if it contains information essential to the proper decoding or understanding of the data. An annotation can be *valuated*, in which case its value is an MP Tree. Finally, a scope qualifier specifies the extent

of the annotation's application within the MP Tree.

- Dictionaries – MP supports collections of definitions for annotations and mathematical symbols (operators and symbolic constants) in *dictionaries*. Dictionaries address the problem of semantic data integration by defining standardized representation(s) and semantics for mathematical objects so that an object's meaning is preserved in any exchange. A dictionary is identified within a packet through a dictionary tag field. Applications that communicate according to definitions provided in dictionaries do not need to have direct knowledge of each other.
- Optimizations – MP uses *prototypes* to minimize the amount of overhead type information that must be transmitted with composite objects that have a homogeneous format. A vector of floating point numbers is a good example, but the data could be more structured - a vector of complex or rational numbers, or a polynomial or ideal. We can take advantage of this pattern by using a **prototype** annotation whose value is an MP Tree specifying the structure and type of the data to be transmitted.

In addition to defining a binary mathematical expression encoding, MP also provides data communication mechanisms. Applications send and receive data as *messages*, each containing an MP Tree which is (typically) created by calling routines from the MP Application Programming Interface. An application communicates with other applications through an *MP link*, which is simply an abstraction of an underlying data transport mechanism that is bound to the link at the time of its creation.

MP has been implemented as a library of C routines using a layered approach as shown in Figure ???. This implementation integrates the representation and data communication aspects of MP.

Layer	Library Function
Packet	MP_Put<type>Packet() MP_Get<type>Packet()
Data	IMP_Put<type>() IMP_Get<type>()
Byte	IMP_PutLong(), IMP_PutBytes() IMP_GetLong(), IMP_GetBytes()
Buffer	Abstract Device Interface
-----	-----
Device	Transport (file, socket, HTTP, MCP, ToolBus, PVM/MPI, ...)

Figure 2: MP-C 1.1.3 implementation layers

An OO redesign and re-implementation of MP is now presented.

## 4 MP Class Library Design

Based on substantial work and experience leading to the C-coded MP-1.1.3 library, we are now redesigning MP using object technology. The effort is motivated by the following goals:

- To make the mathematical data representation and the data communication components of MP independent and usable separately or in combination.
- To create an object-oriented MP class library better able to support diverse distributed/parallel applications.
- To make the MP representation directly usable within IAMC.

Thus, the object-oriented MP class library will consist of two modules: a *representation module* and a *communication module*. A Java implementation can go a long way in these directions. Since the IAMC project will use Java to implement a prototype server and client, the MP data representation module will be directly useful in the IAMC project. Here we focus on the MP data representation module which is responsible for the encoding and decoding of a mathematical expression using the MP format.

- *Encoding* means *serializing* a mathematical expression into a linear sequence of bytes ready to be transported through the network.

$$(+ x y) \implies \text{byte array}$$

- *Decoding* means *deserializing* a sequence of bytes and recovering the mathematical expression.

$$(+ x y) \longleftarrow \text{byte array}$$

The overall strategy (Figure ??) is to employ an annotated parse tree object whose nodes know how to encode/decode themselves. There are Data (leaf) nodes and Operator (internal) nodes. Example node types are ArbInt, Operator, Identifier, Real64, and String.

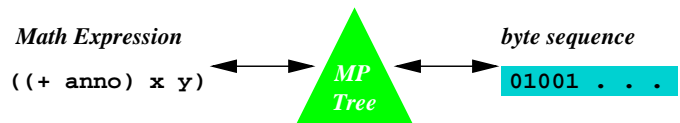


Figure 3: Encode/Decode Strategy

Our object-oriented design for the MP representation module identifies these major classes:

**Message** — the MP message object

**SerialBuffer** — the serialized message buffer object

**Tree** — the annotated parse tree object

**Node** — the superclass of **dataNode**, **operatorNode**, and **protoOpNode**

**Header** — the packet header superclass for **NodeHeader** and **AnnotationHeader**

Figure ?? shows the overview of the class design for the MP data representation module. Solid arrows indicate component (has-a) relations and hollow arrows indicate class extension.

Following the CRC (*Classname-Responsibilities-Collaborators*) design method, we have produced descriptions for the responsibilities and collaborators of each class. Avoiding details, we present the descriptions for **Message**, **Tree**, and **SerialBuffer**.

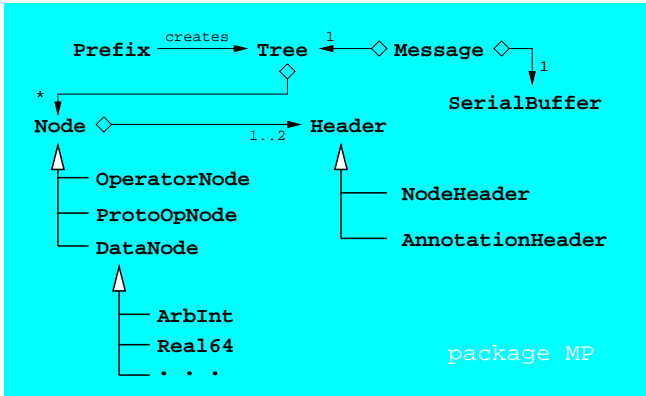


Figure 4: MP Representation Module

## The MP Message class

A `Message` object has the following responsibilities:

1. initialize a `Message` object given an `InputStream` (to read a sequence of bytes which is the serialized MP message), holding the serial encoding of the message in an internal `SerialBuffer`
2. initialize a `Message` object given a `Tree` object
3. serialize the message to a `SerialBuffer` or to a given `OutputStream`
4. deserialize the message to a string in prefix (or infix) notation.
5. provide the byte count of the serialized message

and it collaborates with the `Tree`, `Node`, and `SerialBuffer` classes.

Internally, a `Message` object contains both an MP `Tree` object and a `SerialBuffer` object. The `Tree` object supports the encoding (serialization) of data and the `SerialBuffer` object the decoding (deserialization) of data. Both rely on the `Node` objects to do the actual encoding and decoding.

An application program instantiates a `Message` object, `msg`, either with a `Tree` object or with a byte array. Thus, just after instantiation, `msg` contains internally either a `Tree` or a `SerialBuffer`.

When a `msg.serialize(outStream)` operation is called, for example, `msg` asks the `SerialBuffer` object to produce output, if it is there. Otherwise, `msg` first serializes the `Tree` object into its `SerialBuffer`.

When a `msg.deserialize()` operation is called, the internal `Tree` object is created if it does not already exist. The constants `Message.prefix` and `Message.infix` indicate the target notation for deserialization, with prefix notation the default.

## The Annotated Tree class

A `Tree` object has the following responsibilities:

1. retrieve one node at a time from the tree in preorder
2. build the message tree, adding one node at a time, in preorder
3. substitute a particular node with a given node
4. retrieve a particular node given its preorder sequence number
5. find the first node in preorder that matches the given node content and type
6. produce a byte-array serial representation
7. display itself (data stored) in preorder

and it collaborates with classes `Message`, `Node`, and `SerialBuffer`.

## The SerialBuffer class

The `SerialBuffer` class has the following responsibilities:

1. store a serialized MP encoding of a message (a byte array)
2. build up a serialized MP encoding of a message a chunk at a time
3. retrieve bytes of the message (one node at a time) in sequence

4. provide message complete indicator
5. decode the buffer into a `Tree`

and it collaborates with the `Message`, `Node`, and `Tree` classes.

## 5 Prefix Notation Conversion

To make applications easier, the MP package also supplies the ability to parse and output mathematical expressions in prefix notation, annotated with semantic information.

The `Prefix` class encapsulates the parsing and the display operations for prefix expressions. The function `prefixToMP` takes a prefix expression, parses it, and creates a `Tree` object. The function `prefixDisplay` takes a `Tree` object and produces output in prefix notation.

## 6 MP in IAMC

As stated, the representation module of the MP class library is designed to be used independently. Our first application is in the IAMC system (Figure ??) for making mathematical computation accessible and interoperable over the Internet.

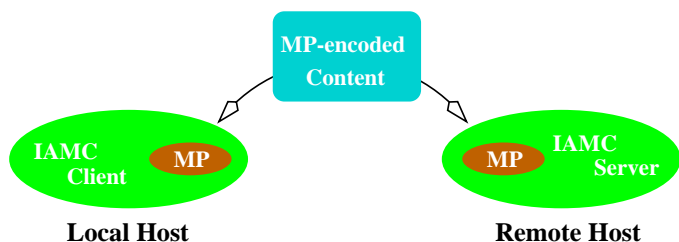


Figure 5: Application of MP in IAMC

IAMC employs MCP for linking clients and servers. An MCP message uses a header-body format similar to HTTP and can transmit a message containing an MP-encoded mathematical expression with the `x-math-MP` content type. Here is a sample MCP message:

```

Response Computation C3 100 OK
Content-type: application/x-math-MP
Content-length: 26
  
```

```

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

Such MP-encoded results can be stored in a file (`result.mp`) to be used later or transmitted to another compute engine. The MP data representation module can help to encode/decode MP messages and convert the messages into infix/prefix expression strings. The full Java API for the MP class library is under development.

## 7 Future Work

The Java MP class library supplies the fundamental classes needed to implement the most recent specification of MP [?]. The next step is to build on these classes to provide functionality similar to that in the C library [?, ?]. An important difference, though, is that with OO technology we will greatly improve on the design and provide more useful (and usable) functionality. In particular, we are interested in creating classes for common mathematical objects and to explore combining (de)serializing methods with MP's prototypes, which have proven to be a powerful mechanism for providing "self-describing" data [?].

A separate project will build additional class libraries to support conversion between expression in the MP format and MathML and  $\text{\LaTeX}$ .

## References

- [1] J. Abbott, A. Diaz, and R. S. Sutor, "A Report on OpenMath," ACM SIGSAM Bulletin, pp. 21-24, March 1996.
- [2] O. Bachmann, S. Gray, and H. Schönemann, "A Proposal for Syntactic Data Integration for Math Protocols," Proceedings, International Symposium on Parallel Symbolic Computation (PASCO'97), pp.

- 165-176, Hawaii, USA, July 1997, ACM Press.
- [3] O. Bachmann, S. Gray, and H. Schönemann, "A Framework for Distributed Polynomial Systems Based on MP," Proceedings, International Symposium on Symbolic and Algebraic Computation (ISSAC'96), Zurich, Switzerland, July 1996, ACM Press.
- [4] O. Bachmann, H. Schönemann, and A. Sorgatz. "Connecting MUPAD and Singular with MP," *MapleTech*, 5(2), Birkhäuser Boston, 1999.
- [5] S. Dalmas, M. Gaëtano, and S. Watt, "An OpenMath 1.0 Implementation," Proceedings, ISSAC'97, ACM Press, pp. 241-248, 1997.
- [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] S. Gray "MP: A Protocol for the Efficient Exchange of Mathematical Data" PhD Dissertation, Kent State University, December 1998.
- [8] 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.
- [9] 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
- [10] P. Ion and R. Miner, ed., *Mathematical Markup Language*, W3C Working Draft, Jan. 6 1998.
- [11] 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.
- [12] P. S. Wang, "Design and Protocol for Internet Accessible Mathematical Computation," Proceedings, ISSAC'99, Vancouver, BC, Canada July 1999.