Article Submitted to Science in China (Series F)

The Internet Accessible Mathematical Computation Framework

Paul Wang*†[1], Simon Gray[2], Norbert Kajler[3], Dongdai Lin[4], Weidong Liao[1] and Xiao Zou[1]

[1] *ICM, Kent State Univ., Kent, Ohio 44242, U.S.A.*

[2] *Ashland Univ., Ashland, Ohio, U.S.A.*

[3] *Ecole des Mines de Paris, 60 bd. St-Michel, 75006 Paris, France*

[4] *State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China*

Abstract

The *Internet Accessible Mathematical Computation* (IAMC) *framework* aims to make it easy to supply mathematical computing powers over the Internet/Web. The protocol-based IAMC framework enables developers to create interoperable clients and servers easily and independently. Presented are conceptual and experimental work on the IAMC framework architecture and major components: the Mathematical Computation Protocol (MCP), a client prototype (*Dragonfly*), a server prototype (*Starfish*), a mathematical encoding converter (XMEC), and an open mathematical compute engine interface (OMEI).

KEYWORDS: Internet Accessible Mathematical Computation, Mathematical Computation Protocol, Open Mathematical Engine Interface, MathML, OpenMath

1. Background

Mathematical computing is inevitably becoming distributed over the Internet – for easy distribution of mathematical materials; to make specialized computations widely accessible; to allow easy interoperability; and to aggregate functionalities from different systems. Indeed, making mathematical communication easy over the Internet has *many* potential applications. While various methods have been used to display mathematical formulas in Web pages and to make

†E-mails: {wliao, xzou, pwang}@cs.kent.edu, sgray@ashland.edu, kajler@paris.ensmp.fr, ddlin@is.ac.cn

*Work reported herein has been supported in part by the National Science Foundation under Grants CCR-9721343 and CCR-0201772 and in part by the Ohio Board of Regents Computer Science Enhancement Funds.

simple mathematical computations accessible via CGI programs or X Windows (ICM live demos, 1999-2001), a general and effective architecture/framework for producing and delivering active mathematical content is still the subject of research and development.

Investigators at the W3 Consortium and elsewhere are working to make *publishing* mathematical materials on the Web easy. *MathML* (Ausbrooks et al., 2000) defines an SGML language for markup of mathematical expressions with support for both presentation (display layout) and content (computation semantics).

The IBM digital publishing group has released the experimental *Techexplorer* (Dooley, 1998), a Web browser plug-in that dynamically formats and displays documents containing scientific and mathematical expressions coded in TEX or LATEX. Some MathML is also supported. Techexplorer also allows a user to send expressions to a fixed compute server for evaluation. *MathType* (Design Science, Inc., 2000) supports interactive creation of mathematical notations for web pages and documents. The same company also offers *WebEQ* that provides a Java applet to display WebTeX and MathML in a browser. The W3 Consortium's *Amaya* Web browser demonstrates a prototype implementation of MathML which allows users to browse and edit Web pages containing mathematical expressions (Vatton and Quint, 2000). Together with the rest of the Web page, these expressions are manipulated through a WYSIWYG interface. The increasing acceptance and software support for MathML were evident at the recent MathML International Conference (2000).

JavaMath (Solomon and Struble, 1998) is a free software package enabling Java-based mathematical programs to use the computational capabilities of existing compute engines. JavaMath can be used for stand-alone applications and for construction of Internet based client-server systems and Web pages. Different from JavaMath, the IAMC *framework* provides a general infrastructure for building Internet/Web-based mathematical computation and education systems. The framework is programming language independent and based on a standard protocol. The framework also follows an open programming interface specification for mathematical compute engines.

IAMC

Mathematical content viewing on a Web page is static. On the Internet, end-users and applications can make good use of *dynamic access to mathematical computing*. "Internet Accessible Mathematical Computation" has been the subject of the *1999 IAMC Workshop* (part of ISSAC'99) and the *2001 IAMC Workshop* (part of ISSAC'01). The full-day workshops underscore the continued interest in making mathematical information and computation easily available in the modern communication age (Linton and Salomom, 1999; Weber and Küchlin, 1999). For more background and related activities, please refer to the IAMC homepage (1999-2001), the online Proceedings of both workshops : IAMC Workshop (1999) and IAMC Workshop (2001) and those of the Future of Mathematical Communication Workshop (1999).

An ongoing research focus at the Institute for Computational Mathematics (ICM/Kent) has been on the communication of semantically rich mathematical data in a distributed, heterogeneous environment. An earlier project explored the efficient encoding and transmission of mathematical data among heterogeneous compute engines (Gray et al., 1994, 1998, 1999).

More recent work has focused on the design and implementation of the *IAMC framework* (Wu, 1998; Wang, 1999).

The IAMC framework

The IAMC framework aims to make it easy to connect and interoperate heterogeneous mathematical clients and servers, to support both interactive and transparent access to mathematical computation on the Internet/Web through a suitable protocol, and to provide customizable software prototypes and libraries to make setting up Internet-based mathematical services easy.

The following are some potential applications of IAMC.

- From the end-user perspective: easily accessing remote mathematical computations and databases, including access to highly specialized computing services; the possibility to exchange and further process computational results among different compute servers.
- From the developer perspective: making math-oriented data and services easily and widely accessible on the Internet in many contexts (Web vs Email, small vs large bandwidth, etc.); simplify the use of emerging technologies and standards such as MathML (Ausbrooks et al., 2000), OpenMath (Abbott et al., 1996), UDDI (2000), SOAP (2000), WSDL (2001), etc.
- More generally: the possibility to speedup building and adoption of innovative mathematical software under the form of "Web services" in the areas of mathematical research, engineering, computer-aided mathematical education and distance learning.

In terms of design (and implementation), our main goal with the IAMC framework is to provide a *coherent set of customizable software components and libraries* that can be used either

1. *together*, for setting up a specific distributed mathematical software system;
2. or *independently*, in other mathematical software applications.

Each component of the IAMC framework is explicitly designed for independent re-use outside of the framework. Furthermore, the IAMC framework architecture is kept flexible so that it can be extended and tailored as needed, to be used as a testbed for research or as a platform for software development.

This way, we hope to achieve maximum *flexibility*, for ourselves and others, in the continued development of Internet-based mathematical services.

Reported here are advances in the design, architecture, protocol, data encoding, and software prototyping for the distributed IAMC framework.

2. IAMC Framework Architecture

The IAMC framework is designed to make mathematical computing easily accessible on the Web/Internet (Wang, 1998, 1999). It follows a multi-layer architecture to gain performance and

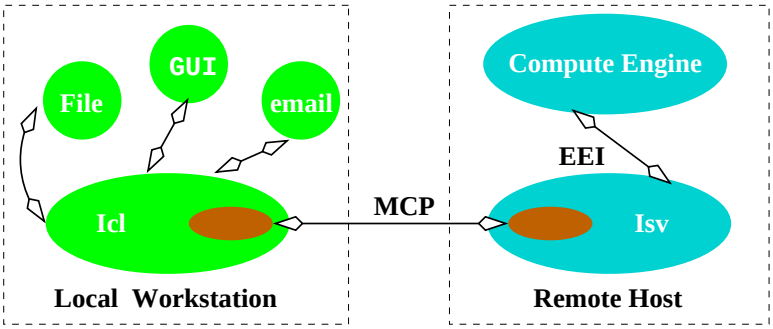

Figure 1: IAMC Architecture

scalability. This architecture can also be expanded easily to meet the requirements for Web-based mathematical computation and education services. Figure 1 shows the architecture of the IAMC framework. The IAMC framework consists of the following components:

1. IAMC client (Icl) — An end-user agent for accessing services provided by any IAMC server. Dragonfly (see Section 6.1) is a prototype Icl implemented in Java.
2. IAMC server (Isv) — A program to provide mathematical computation powers through the MCP protocol. An Isv may or may not employ an external compute engine to perform mathematical computations. Starfish (see Section 6.2) is a prototype Isv implemented in Java.
3. Mathematical Computation Protocol (MCP) — IAMC clients and servers are connected by the Mathematical Computation Protocol (see Section 3). MCP aims to be a simple and efficient request-response protocol to support both one-time transactions and interactive sessions. MCP supports automated computation requests from applications that transparently contact servers for mathematical computation and/or information. For example, an HTTP request may trigger server-side actions that obtain mathematical results from an Isv via MCP.
4. Mathematical Data Encoding — Standard and user-defined mathematical data encodings can be used within MCP (see Section 4). Standard formats (such as MathML, MP, or OpenMath) allow heterogeneous mathematical programs to interoperate. The IAMC framework allows plug-in format converters on the client and the server side, to support additional formats. Automatic negotiation between the Icl and Isv at the beginning of a computation session determines the encoding(s) used. The IAMC framework also provides XMEC, a versatile mathematical encoding converter that has wide applicability (see Section 4).
5. External Engine Interface (EEI) — The EEI is the boundary between an IAMC server and an external compute engine. The implementation of the EEI is an engine-specific driver that handles translation between the format of the request supplied by the Icl and the format of the compute engine (and similarly for responses from the engine). To

simplify the construction of drivers, we have defined as part of the IAMC framework the *Open Mathematical Engine Interface* (OMEI), an API specification for interfaces to mathematical compute engines (see Section 5).

Dragonfly (Liao and Wang, 1998), *Starfish* (Liao and Wang, 2000), *JMP*, an OO implementation of MP (Gray et al., 1999; Tong, 2000), and the *eXtensible Mathematical Encoding Converter* (XMEC) have been implemented in Java. The Dragonfly-Starfish approach makes interactive mathematical computations accessible and interoperable over the Internet; XMEC facilitates the development of practical IAMC systems.

The prototype Icl and Isv were built to test the effectiveness of the IAMC architecture and evaluate/improve the MCP protocol. JMP and XMEC are independent of the IAMC framework and could easily be used in other applications.

3. The Mathematical Computation Protocol (MCP)

For connecting IAMC clients to servers, we need a well-designed protocol that makes accessing mathematical computation on the Internet effective and efficient. MCP, the Mathematical Computation Protocol, aims to be the core of such a 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.
- 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 control.
- Being simple, effective, and extensible.

As described in its original design (Wang, 1999), MCP uses HTTP-style requests and responses to support mathematical computation sessions. Specific MCP requests belong to different classes that contain request methods to support well-defined operations. Following the OO philosophy, each MCP request class defines a set of methods that can be invoked. Standard MCP request classes include:

- `Initialization` — The initialization class supports session creation and configuration right after client-server connection.
- `Control` — The control class supplies MCP protocol control and management methods for both the client and server side.
- `Computation` — A class of server-side operations to perform application supported computations.
- `Dialog` — A class of client-side methods to solicit information from the end user.

As in HTTP, status codes and strings are used in responses to indicate error and exceptional conditions.

4. Mathematical Data Encoding

The MCP protocol allows any content type to be transported within the MCP message envelope. In principle, IAMC servers and clients can use any mathematical data encoding they wish. In practice, clients and servers should use standardized and widely understood mathematical representations.

It is reasonable to assume, at this point, that MathML will become a standard. Hence, MCP has some built-in support for MathML. It has a converter module MathML_MP that can convert between MathML and MP. Because MP is a very compact binary encoding, it can perform *MathML compression* before network transmission. For the same reason, MP can be used in place of MathML when data size and/or the frequency of data exchanges creates a serious performance bottleneck (Avitzur et al., 1995).

MathML defines the default semantics of content encoding elements. The definitions are in OpenMath format. For a content element, the `definitionURL` and `encoding` attributes can specify a particular *content dictionary* (CD) and its definition format, respectively. Further, the `<semantics>` tag can be used to attach CD or other semantics mapping information to content-encoded expressions. MCP is designed to allow any math encoding using any CDs. It supports Isv-Icl negotiation on the CD set to use via the `dictionary-map` method in the `initialization` class. Adding/deleting from the CD set during the computation session is also supported. Internally, MCP stores the CD set in a *CD map*, so integer indexes can be used in place of CD tags for data compression purposes.

The eXtensible Mathematical Encoding Converter (XMEC)

A typical data flow through the IAMC framework is:

1. The user enters input in infix notation.
2. The Icl parses the user input and translates it into MathML content encoding.
3. MathML content data is sent through MCP with automatic MP compression/decompression.
4. The Isv sends MathML content encoding to the compute engine through the engine's OMEI driver.
5. The OMEI driver converts the MathML content encoding into the engine's specific syntax and representation, and submits it to the engine for evaluation.
6. The OMEI driver retrieves the result from the compute engine and converts it into MathML content and/or presentation encoding.
7. The Isv receives the MathML content and/or MathML presentation result from the engine driver.
8. The Isv sends the result back to the Icl via MCP.
9. The Icl GUI gives the MathML presentation code to a renderer for display.

This typical scenario clearly illustrates the need for mathematical data format conversions in a distributed computation environment, especially when heterogeneous components are involved. As part of the IAMC framework, we have designed and implemented XMEC in Java

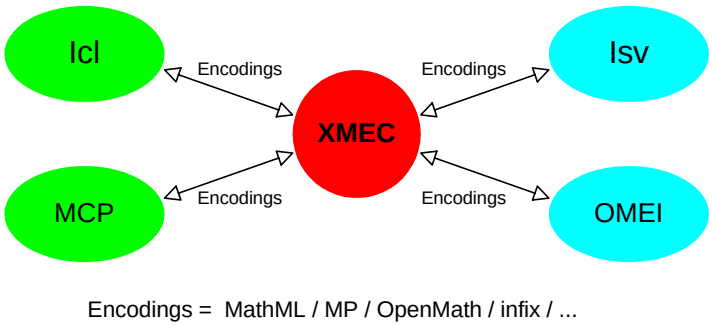

Figure 2: XMEC in the IAMC Framework

(see Figure 2), based on JMP, an OO implementation of MP in Java (Gray et al., 1999; Tong, 2000).

XMEC provides native support for translations among infix notation, MathML, and MP. As a side benefit, experiments with MathML-MP conversions have shown good conversion speeds and significant data compression. It may be advantageous to convert large MathML results to MP before sending them through the network.

XMEC plays an important role within the IAMC framework. Both the MCP library and OMEI drivers (see Section 5) can use XMEC as a built-in component. However, XMEC can also be used outside the IAMC framework as a standalone mathematical encoding converter (or as part of another software framework). Its design makes it suitable whenever there is a need to convert from one encoding to another, either "by hand" (on end-user request) or "on the fly" as part of a Web-based software architecture. XMEC has been demonstrated at the IAMC'2001 Workshop (Zou, 2001) and is available from the IAMC framework site (1998-2001).

MathML/Graph

In addition to expressions and formulas, IAMC also supports plotting of mathematical curves and surfaces. To represent graphing requests and results, we devised an XML-based format called *MathML/Graph* which can be viewed as a *graphing extension* to MathML and can be used within IAMC or independently.

With XML we define several new tags, including `<mathGraph>`, for representing a plotted graph; `<mathPlot>` for issuing plotting requests; `<range>` and `<coordinates>`.

Generally, a curve or surface may involve many points and the representation can become extremely large. As suggested in Avitzur et al. (1995), binary encoding and data compression techniques can significantly reduce the size of data to be transmitted and the overall encoding + transmission + decoding time. Within the IAMC framework, MathML/Graph can (should) be compressed by the MathML_MP converter bacause MP is particularly efficient in representing floating-point arrays.

5. The Open Mathematical Engine Interface (OMEI)

To make the IAMC framework valuable, many useful computational services must be made available. Hence, it is important to be able to create IAMC services easily. The generic Starfish (see Section 6.2) runs on any Java platform and goes a long way in this direction. But connecting to external computation engines must also be made easier. For example, JavaMath (Solomon and Struble, 1998) supplies a uniform API to make it easy for Java-based programs to access existing compute engines.

We have the initial specification of the *Open Mathematical Engine Interface* (OMEI) (Liao et al., 2001) to govern how IAMC servers and external compute engines interface. Making different compute engines interoperate is a challenge. The semantics and behavior of most compute engines are not even specified completely (Fateman, 1996). Work on OMEI and the IAMC framework must proceed in this less than ideal context.

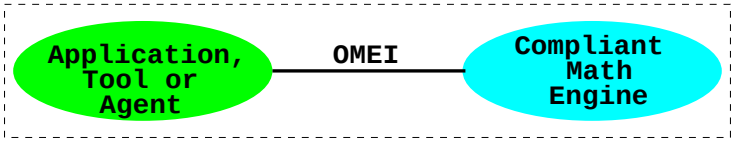

Figure 3: Interfacing Using OMEI

OMEI (Liao and Wang, 2000) is language independent and modeled after the standard database programming interface CLI (Call-Level Interface). OMEI specifies a standard API to create connections, obtain capabilities, retrieve help/documentation, formulate and execute commands, receive results, etc., for mathematical compute engines (see Figure 3). Developers can map OMEI to different programming languages and implement *OMEI drivers* to provide access to different applications. The OMEI driver is a software module that provides a uniform API to different mathematical engines (see Figure 4). The engine end of a driver needs to be customized for the compute engine to which it is to be attached. A compute engine designed to go online may support OMEI directly, or may bypass the OMEI driver.

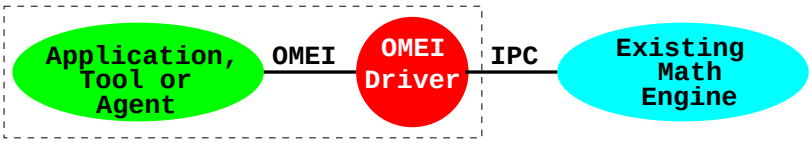

Figure 4: Interfacing with an OMEI Driver

The OMEI specification plays a very important role in the IAMC framework; it governs and simplifies how IAMC servers interact with compute engines. To respond to client requests and perform computations, the IAMC servers have to load OMEI drivers first.

OMEI was specified by studying existing engines. It establishes several categories of function descriptors to encapsulate properties of compute engines and thus gives a unified programming-level interface for heterogeneous mathematical engines. The OMEI implementation prototypes (see Section 6.3) supports several categories of operations for server-engine interaction:

1. Connecting to/disconnecting from a compute engine.
2. Querying engine capabilities.
3. Creating commands.
4. Executing commands.

Connecting and Disconnecting

Before the computational power of a compute engine can be exploited, a *communication channel* must be established. This channel is vital for server-engine interaction because it is how the commands are transmitted and the results obtained. The communication channel could be a simple inter-process communication (IPC) pipe or socket, or a network link based on a communication protocol. Usually, a channel must also contain the context for the specific server-engine pair. Such context information includes the current command, the status of its completion, the result after the command execution is completed, and the error information if the command cannot be executed.

In OMEI, the term *Connection* refers to the communication channel and the term *Connection Handle* refers to the connection context. Before a connection is established, a connection handle must be allocated. This connection handle can be freed after the connection is closed.

Querying Engine Capabilities

Once a connection is established, the *application* is ready to interact with the compute engine. Before submitting the actual computation requests to the engine, the application can *inspect* the computational capabilities supported by the compute engine.

Four types of engine capability information can be requested:

- A *CanDo list* that tells what computations the engine can provide for this specific connection.
- *Help* information for each individual command.
- *Command template* that gives the syntax of an individual command.
- *Mathematical encodings* supported.

Creating Commands

A command can be created in three different ways: by supplying a string for the command, by giving a file containing a mathematical expression, or by making several OMEI calls to specify the command name and arguments. In the first and second approaches, the command can be either a literal string representing a native command for the engine, or math-encoded in a format such as MathML, OpenMath or MP. In the third approach, several OMEI calls are used to specify the command name and arguments.

Executing Commands

Once a command has been created, it can be sent to the engine for execution. The execution can be either synchronous or asynchronous, and it may be interrupted or cancelled before it is completed. As a result, a group of function descriptors has been created to launch and coordinate the command execution.

6. Software Prototype for IAMC Framework

In this section, we will introduce our software prototyping for the distributed IAMC framework. They were built to test the effectiveness of the IAMC architecture and evaluate/improve the MCP protocol and OMEI specification.

6.1. Client Prototype: Dragonfly

Dragonfly (Liao and Wang, 1998) is a prototype IAMC client implemented in Java. Distinct from other front ends, Dragonfly is a protocol-compliant, full-featured graphical user interface for accessing any IAMC compliant server. Dragonfly can:

- Connect to and communicate with any user-specified IAMC server via the MCP protocol.
- Obtain capability and usage documentation from the IAMC server and make them available to the user, including presenting command templates from the server to the user when requested.
- Receive computational, control and help commands from the user, encode and send them to the server.
- Receive and decode results from the server.
- Display mathematical symbols and expressions in textbook-like fashion and allow selection of subexpressions with a mouse.
- Plot mathematical curves and surfaces.
- Save mathematical results in files under well-defined formats such as MP, MathML, and OpenMath.
- Display server dialog and relay user data thus obtained back to the server.

Furthermore, Dragonfly is designed for extensibility allowing easy addition of features and functionalities. Built on the Java 2 platform, Dragonfly employs:

- Swing to implement the GUI and the HTML-based help facility.
- Java 2D to support 2-dimensional and 3-dimensional mathematical function plotting and manipulation.
- WebEQ to render mathematical expressions in MathML presentation code.
- XML to help encode/decode graphing data.

The current version of Dragonfly connects to one server at a time. To connect to multiple servers, the user has to open multiple Dragonfly instances. The user can use the "Copy/Paste" mechanism to exchange information among different Dragonfly instances. The ability to connect to multiple servers and issue concurrent computations is important and is planned for a future version.

6.2. Server Prototype: Starfish

An IAMC server (Isv) uses MCP which has the IANA registered user port

```
mcp-port        3558/tcp   MCP user port
mcp-port        3558/udp   MCP user port
```

Once connected, the Icl and Isv conduct business using MCP. When the computation session is done, the Isv terminates. An IAMC server has the following functional requirements:

- Be MCP compliant and easily customizable, for example, by allowing plug-in modules for extensibility.
- Maintain and manage computation sessions.
- Launch/control external compute engines dynamically.
- Support synchronous/asynchronous calls, callbacks, and client-generated interrupts.
- Handle MCP computation requests either synchronously or asynchronously.

Starfish is our IAMC server prototype implemented in Java. Through Java multi-threading, Starfish can support multiple concurrent Icl connections. Starfish uses a property file to identify the engines available to it and to locate and dynamically load *OMEI drivers* (see Section 6.3) for particular services. This provides great flexibility, allowing Starfish to dynamically bind to a compute engine based on the nature of the client request. Starfish also has an administrative GUI for configuration and activities monitoring.

6.3. OMEI Driver Prototypes in Java

Starfish loads OMEI-compliant drivers implemented in Java to access external compute engines. We call the Java definition of OMEI *Java Mathematical Engine Interface* (JMEI). To interface a specific compute engine, JMEI can be applied in these three approaches:

1. To use inter-process communication (IPC) to access the mathematical engine, a JMEI driver may utilize the `java.lang.Process` and `java.lang.Runtime` facilities.
2. To use procedure calls to access the engine, a JMEI driver may employ the Java Native Interface (JNI) scheme. This approach is feasible only if the compute engine provides an API for a host language that JNI supports.
3. To use networking to access a remote engine, such as Mathematica, a JMEI driver can use a 2-layer approach as shown in Figure 5. In this approach, the JMEI driver becomes a front end to the engine-defined networking interface.

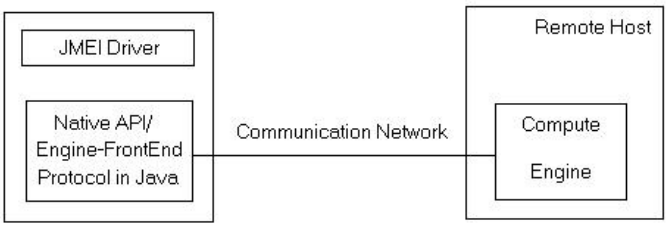

Figure 5: Implement a Network-Enabled JMEI Driver

The IPC approach is the most general, and consequently the most complicated to implement. If the computational package only supports a runtime system for *interactive* use – that is, without API support for any mainstream programming language – the JMEI driver must be IPC-based. In this case, JMEI employs a two-way pipe between itself and the compute engine.

The complication for the IPC approach lies in the synchronization between the process's input channel and output channel. To read the execution results, the beginning and end of the result for each command must be identified correctly. This may become quite involved if we take into consideration that the result may be an error message. The solution is often engine-specific. This is greatly simplified if the compute engine can output results and errors in a structured format such as MathML.

For the prototype IAMC framework, we have implemented JMEI drivers for Maxima with IPC approach and for Mathematica by using J/Link. The Mathematica JMEI driver is network-enabled to allow an application to interface with Mathematica running on a romote host. OMEI has also been applied to interface ELIMINO (Wu et al., 2002) with Dragonfly providing this specialized mathematical tool with a local and remote GUI.

7. Conclusions and Future Work

The IAMC framework is an effort to establish a protocol-based, platform, programming language, and mathematical encoding independent, solution for serving mathematical computation over the Web/Internet. In the Framework, connectivity between client and server is defined by the *Mathematical Computation Protocol* (MCP) while connectivity between IAMC servers and external compute engines follows the *Open Mathematical Engine Interface* API specification. The design helps make the IAMC framework open, effective, and efficient. Some parts, such as OMEI, Dragonfly, JMP, and XMEC, are of interest in applications independent of the IAMC Framework

Our group at ICM demonstrated Dragonfly, Starfish, and XMEC at the IAMC Workshop (2001). Work on the specification and library implementation of MCP and the XML implementation of MathML/Graph is ongoing. We are also considering a Web-based search engine for end-users to locate IAMC servers that support particular computations and converting Dragonfly into a valid plug-in to be used with Netscape.

One of our goals is to create a coherent collection of reusable Java components covering all aspects of an effective IAMC implementation based on existing and emerging Web technologies (e.g., XML, MathML, MP, OpenMath and for the close future UDDI, SOAP, etc.) Those components are intended to form a flexible framework usable as a testbed for research or a platform for software development. Each component is also intended to be separately re-usable outside of the framework. Our prototype implementation of the IAMC framework, including MCP, Dragonfly, Starfish, OMEI, and XMEC, represents a first step in this direction.

We hope to collaborate with more people in different parts of the world. The goal is to make mathematical computations easily accessible in many contexts world-wide. Our progress can be tracked on the IAMC framework site (1998-2001).

References

ICM, SymbolicNet live demos (accessible from `http://www.symbolicnet.org/systems/-demos.html`), (2000).

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. Available from: `http://www.w3.org/TR`, (2000).

S. Dooley, Coordinating Mathematical Content and Presentation Markup in Interactive Mathematical Documents, *ISSAC'98*, O. Gloor, Ed., (1998). ACM Press, 54–61.

Design Science, Inc., MathType and WebEQ (software available from `http://www.mathtype.com`), (2000).

I. Vatton, V. Quint, Editing MathML on the Web with Amaya, *In MathML International Conference (2000)*, (2000).

MathML International Conference. Procs. available from `http://www.-mathmlconference.org`, (2000).

A. Solomon, C. A. Struble, JavaMath (software available from `http://javamath-.sourceforge.net`), (1998).

S. Linton, A. Solomon, GAP, OpenMath, and MCP. In the online Proc. of IAMC Workshop (1999), (1999).

A. Weber, W. Küchlin, A Framework for Internet Accessible Software Components for Scientific Computing. In IAMC Workshop (1999), (1999).

IAMC homepage : `http://icm.mcs.kent.edu/research/iamc`, (1999-2001).

IAMC'99 workshop. Procs. available from `http://icm.mcs.kent.edu/research/iamc99proceedings.html`, (1999).

IAMC'01 workshop. Procs. available from `http://icm.mcs.kent.edu/research/iamc01proceedings.html`, (2001).

Future of Mathematical Communication workshop. Procs. available from `http://msri.org/calendar/workshops/9900/Future_of_Math_Communications`, (1999).

S. Gray, N. Kajler, P. Wang, MP: A Protocol for Efficient Exchange of Mathematical Expressions, *ISSAC'94*, M. Giesbrecht, Ed., (1994). ACM Press, 330–335.

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

S. Gray, L. Tong, P. Wang, The MP Encoding for Distributed Mathematical Computations: An Object-oriented Design and Implementation, *Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, H.R. Arabnia, Ed., (1999). CSREA Press, 2084–2090.

W. Wu, *Experiments with Internet Accessible Mathematical Computation.* Master's Thesis, Dept. of Math. and Computer Science, Kent State Univ., ICM Technical Report 199805-0003 (available from `http://icm.mcs.kent.edu/reports`), (1998).

P. Wang, Design and Protocol for Internet Accessible Mathematical Computation, *ISSAC'99*, S. Dooley, Ed., (1999). ACM Press, 291–298.

J. Abbott, A. Diaz, R. S. Sutor, Report on OpenMath, *ACM SIGSAM Bulletin*, (1996). **1**, 21–24.

UDDI (Universal Description, Discovery and Integration). `http://www.uddi.org`, (2000).

SOAP (Simple Object Access Protocol). `http://www.w3.org/TR/soap`, (2000).

WSDL (Web Services Description Language). `http://www.w3.org/TR/wsdl`, (2001).

J. M. Purtilo, *A Software Interconnection Technology to Support Specification of Computational Environments.* PhD. Thesis, Dpt. of Computer Science, Univ. of Illinois at Urbana-Champaign, (1986).

D. Arnon, R. Beach, K. McIsaac, C. Waldspurger, CaminoReal: An Interactive Mathematical Notebook, *Intl. Conf. on Electronic Publishing, Document Manipulation, and Typography (EP'88)*, J. C. van Vliet, Ed., (1988). Cambridge Univ. Press, 1–18.

A. Diaz, E. Kaltofen, K. Schmitz, T. Valente, M. Hitz, A. Lobo, P. Smyth, DSC: A System for Distributed Symbolic Computation, *ISSAC'91*, S. Watt, Ed., (1991). ACM Press, 323–332.

N. Kajler, CAS/PI: a Portable and Extensible Interface for Computer Algebra Systems, *ISSAC'92*, P. Wang, Ed., (1992). ACM Press, 376–386.

S. Dalmas, M. Gaëtano, A. Sausse, Distributed Computer Algebra: the Central Control Approach, *1st Intl. Symp. on Parallel Symbolic Computation (PASCO'94)*, H. Hong, Ed., (1994). World Scientific (Lecture Notes Series in Computing, **5**), 104–113.

Y. Doleh, *The Design and Implementation of a System Independent User Interface for an Integrated Scientific Computing Environment.* PhD. Thesis, Dept. of Math. and Computer Science, Kent State Univ., (1995).

J. Dongarra et al., NetSolve, (software available from `http://icl.cs.utk.edu/netsolve`), (2000).

P. Wang, Internet Accessible Mathematical Computation, *3rd Asian Symp. on Computer Mathematics (ASCM'98)*, (1998). 1–13.

W. Liao, P. Wang, Dragonfly: A Java-based IAMC Client Prototype, *4th Asian Symp. on Computer Mathematics (ASCM'2000)*, (1998).

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

L. Tong, *Object Oriented Design and Implementation of MP.* Master's Thesis, Dept. of Math. and Computer Science, Kent State Univ., (2000).

R. Avitzur, O. Bachmann, N. Kajler, From Honest to Intelligent Plotting, *ISSAC'95*, A. H. M. Levelt, Ed., (1995). ACM Press, 32–41.

X. Zou, XMEC: An Extensible Mathematical Encoding Converter. Demo at IAMC'2001 Workshop and ICM Technical Report 200108-0005, Kent State Univ. (available from `http://icm.mcs.kent.edu/reports/`), (2001).

IAMC framework site : `http://icm.mcs.kent.edu/research/iamcproject.html`, (1998-2001).

W. Liao, D. Lin, P. Wang, OMEI: An Open Mathematical Engine Interface, *5th Asian Symp. on Computer Mathematics (ASCM'2001)*, (2001). World Scientific Press (Lecture Notes Series on Computing, **9**), 82–91.

R. Fateman, Symbolic Mathematics System Evaluators, *ISSAC'96*, Y. N. Lakshman, Ed., (1996). ACM Press, 86–94.

Y. Wu, W. Liao, D. Lin, P. Wang, Local and Remote User Interface for ELIMINO through OMEI, *Mathematical Software(ICMS'2002), Proc. Int. Congress of Math. Software*, (2002). World Scientific Press, 411–420.