Open Mathematical Engine Interface and Its Application [*]

W. Liao , D. Lin[†], P. Wang Y. Wu[‡]
Institute of Computational Mathematics
Kent State University
Kent, Ohio 44242, U.S.A.

Abstract

The *Open Mathematical Engine Interface* (OMEI) aims to establish a uniform application programming interface (API) for heterogeneous mathematical computation engines. A common API such as OMEI can make mathematical engines easily accessible by front-ends, tools, and servers. It also enables the development of individual applications that can serve or utilize different engines. The motivation, application framework, specification, and usage scenarios for OMEI are described. The OMEI interface is applied to implement an API for ELIMINO, a symbolic system specialing in polynomial chareristic sets computations. With the OMEI API, ELIMINO becomes a compute engine that is compliant to the *Internet Accessible Mathematical Computation* (IAMC) framework. Thus, ELLIMINO gains the Dragonfly GUI for local and remote users as well as the ability to provide computational services over the Web/Internet.

1 Introduction

The *Open Mathematical Engine Interface* (OMEI) is an application programming interface (API) specification for mathematical compute engines. Following a standard interface specification such as OMEI, applications can access any compliant mathematical engines and vice versa. The concept of OMEI is similar to that of Microsoft's *Open Database Connectivity* (ODBC) which has become the standard for PCs and LANs. Be becoming OMEI compliant, distributed mathematical components, such as front ends, servers, tools can easily interoperate with mathematical engines.

[*]This work has been partly announced in 2001 Asian Symposium on Computer Mathematics and 2002 International Congress on Mathematical Software. Emails: {wliao, pwang}@cs.kent.edu, ddlin@is.ac.cn, wuyw@mmrc.iss.ac.cn

[†]SKLOIS, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China.

[‡]Department of Computer Science and Technology, Tsinghua University, Beijing, China

The motivation for this work comes from several areas. First, it is evident that the demand to make mathematical computing accessible over the Internet is increasing. Cooperating with other institutions worldwide, the Institute for Computational Mathematics (ICM) at Kent State University initiated the *Internet Accessible Mathematical Computation* (IAMC) framework project [8, 9, 17, 18, 19] to provide an infrastructure for bringing mathematical computational and educational services over the Internet. The IAMC framework includes an IAMC client, an IAMC server, and a layered protocol model for connecting IAMC clients and servers effectively and efficiently over the Internet. The computation powers of an existing mathematical compute engine can be made available on the Web/Internet by an IAMC server or other similar programs. Consequently, it is essential to have an easy and systematic way for network servers to interface with compute engines.

The second motivation comes from *distributed mathematical computation* (DMC), an important research area in symbolic and numeric computation. The goal of DMC is to make mathematical computation accessible and interoperable remotely. The DMC architecture generally consists of a user interface, a programming interface, and a mathematical data encoding on top of the communication network/protocol layers. Contributions can be easily seen in the user interface (e.g. CAS/PI [6], SUI [2] and GI/S [25]), and the encoding (e.g. OpenMath [1], MathML [4] and MP [3]) , and the protocol (e.g. MCP [18, 19] and KQML [7]) levels. Nevertheless, the programming interface area requires more investigation. With a well-defined application programming interface, distributed mathematical components, such as front ends, servers, and GUIs, can interoperate with different mathematical engines. OMEI is an effort in this direction.

The third motivation for OMEI comes from the development of new mathematical systems. Generally, a mathematical system contains two main parts: a computation kernel and a user interface. The same kernel can be served by a number of user interfaces designed for different end users—in industry, education, or scientific research, for example. Depending on its purpose, a user interface can be simple and straight-forward or sophisticated and complex. A standard such as OMEI can separate the development of mathematical engines from user interfaces. Therefore, the two can be developed independently and both will be usable with any OMEI compliant components.

The last but not least motivation is from main-stream applications that can use mathematical enhancements. For example, the word processing system Microsoft Word comes with an Equation Editor to embed mathematical expressions into documents. It would be nice to be able to also evaluate such expressions and to plot their graphs. OMEI compliant mathematical engines will be easier to integrate into such systems either as an embedded component or an external process.

APIs for specific mathematical engines, such as Math/Link [13] by Wolfram Research Inc. and Matlab External Interface [14] from MathWorks, exist. They are vendor/engine specific and programming language dependent. Nevertheless, these interfaces provide valuable input and excellent reference for the OMEI effort.

Several other efforts for distributed mathematical computation environment are also noted. JavaMath [5] is proposed as a standard Java API for client-server mathematical computation over Java and Java RMI. OpenXM provides a communication protocol and also a programming paradigm over its protocol layer. Different from JavaMath and OpenXM, OMEI is an abstract programming interface specification that works with any protocol and mathematical encoding. It is language, platform, communication protocol, and mathematical engine independent. Implementations of OMEI are of course written in specific programming languages for their target mathematical engines.

OMEI has been applied in the IAMC framework to build APIs for the general symbolic computation systems MAXIMA and Mathematica, making them IAMC compute engines. As a result, these systems are accessible through the IAMC front-end Dragonfly on the Internet. Here, we focus on describing the OMEI specification and its application for the ELIMINO symbolic computation system.

2 OMEI Overview

The *Open Mathematical Engine Interface* (OMEI) is an application programming interface (API) specification that aims to be general enough to work for most mathematical compute engines. It specifies a set of function prototypes together with their syntax and semantics to give a unified programming level interface for heterogeneous mathematical engines. These prototypes support all operations that are necessary for server-engine interaction, including connecting to/disconnecting from a compute engine, querying engine capabilities, creating and executing commands, and so on.

OMEI can help mathematical software developers achieve the following goals:

- **Engine-Application Interoperability**
  Compute engines following the OMEI specification will be interoperable with any user interface, tools, and applications that employ the interface. Conversely, an application using OMEI to access one compute engine can access any other engine that is OMEI compliant. With help of the OMEI specification, compute engines, user interfaces, and applications can be developed separately and independently and still work together.

- **Network Accessibility**
  An OMEI compliant mathematical engine can readily interface to Starfish, an IAMC server prototype. This means the engine can be used from the Internet via the IAMC framework. The IAMC framework follows a 3-tier architecture that consists of IAMC client, IAMC server, and back-end compute engine. IAMC uses OMEI to connect IAMC servers, such as Starfish, to back end engines.

- **Integration of Heterogeneous Mathematical Systems**
  An application can access and integrate multiple engines by loading multiple OMEI drivers. An integrated compute engine can combine capabilities from other engines using OMEI. A parallel/distributed problem solving environment can be much easier to achieve over the OMEI programming paradigm.

- **3-tier/multi-tier mathematical systems**
  In general, OMEI can make building 3-tier/multi-tier distributed mathematical systems easier by standardizing the engine API.

3 OMEI Specification

OMEI aims to be an interface general enough to work for most mathematical compute engines. It is specified after studying existing mathematical engines and their user/programming interfaces. The following properties were noticed:

- A compute engine may execute some initialization commands before it begins to perform user-requested computations.
- A compute engine normally takes one command at a time. Each command has a terminating character.
- Commands in the form of mathematical expressions are usually encoded in either ad-hoc or standard formats. As an example of standard encoding format, XML/MathML [4] is getting more and more support from mathematical engine developers.
- A compute engine returns a result or an error message for a command. Results may contain text, mathematical expressions, and graphics. The beginning and end of the result are well-defined.
- A compute engine may ask for extra information from the user in order to complete a particular computation.
- A compute engine may keep track of commands and results.
- Help information and documentation can come from the engine or some other sources.
- To conduct computations with a compute engine through its API, an external program usually needs to create and maintain a persistent connection with the engine.
- To manage computation requests to a compute engine through its API, an external program may need to keep track of certain environmental information such as the status of previously submitted requests.
- Before quitting, an external program usually must close the connection and free the allocated resources.

OMEI establishes several categories of function prototypes to encapsulate such properties and thus gives a unified programming-level interface for heterogeneous mathematical engines. These prototypes supports all operations that are necessary for server-engine interaction, including connecting to/disconnecting from a compute engine, querying engine capabilities, creating and executing commands, and so on.

Two handles are used in all these function prototypes: `OMEICONHandle` and `OMEICMDHandle`. They correspond to the execution environment for each connection and command, respectively. Before making connection to a compute engine, the program must allocate a connection handle. Similarly, a command handle must be allocated before a command is created.

Every OMEI call returns an `OMEIRETURN` code, which is an integer representing the execution status for each call: `-1` means an error occurs; `0` means the call is successful. The meaning of other return values depends on the individual call.

3.1 Connect and Disconnect

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 where the commands are issued 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. In OMEI, the term *Connection* refers to the communication channel and the term *Connection Handle* refers to the connection context.

3.2 Query 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 identified: a *CanDo list* that tells what computations the compute 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. Correspondingly, OMEI provides four query operations: `OMEIGETCanDo()`, `OMEIGetHelpOnCommand()`, `OMEIGetTemplateOnCommand()`, and `OMEIGetSupportedEncodings()`.

3.3 Creating Commands

A command can be created in three different ways: by giving a string for the command, by giving a file containing a mathematical expression, or by making several OMEI calls to enter 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,

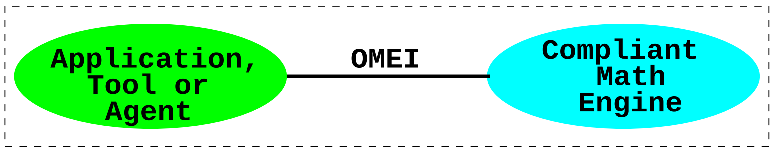


Figure 1: Interfacing Using OMEI

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.

A *command handle* must be allocated before a command is created. The command handle is used to save the execution context for an individual command. From the command handle, we can check the status of a command, fetch the execution result, and interrupt the execution if necessary. Once the command execution is finished and the result is fetched, the command handle can be released to free the related resources.

3.4 Executing Commands

A created command is then forwarded to the engine for execution. OMEI specifies a group of function prototypes to coordinate the command execution. There are five function prototypes in this group: `OMEIExecuteCommand()`, `OMEICancel()`, `OMEIResultEncoding()`, `OMEIGetResult()`, `OMEIPutData()` , and `OMEI-QueryDataType()` ..

4 Applying OMEI

The OMEI specification can be employed to build an integrated computation environment based on existing compute engines. Moreover, it also provides the reference programming user interface for new compute engines. The application framework for OMEI is depicted in Figure 1. Herein the front-end is a mathematical application, a toolkit, or a networking agent such as a server for the Internet/Web, and the OMEI compliant mathematical engine refers to the type of engines that come with an OMEI compliant API.

Interfacing to existing mathematical engines is another story. An existing engine can certainly be reprogrammed to support OMEI. Another approach is to develop a separate *OMEI driver*, a program that implements the OMEI interface for a particular engine in a specific programming language. In this case, the front-end and the OMEI driver run in one process. Depending on the driver implementation, the engine may be internal, external, or remote. Figure 2 shows the application framework in this situation.

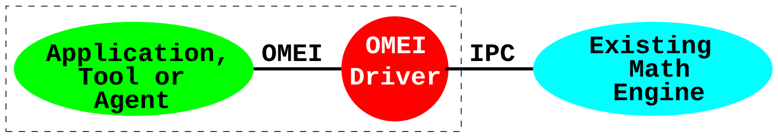


Figure 2: Interfacing with an OMEI Driver

5 OMEI and ELIMINO

ELIMINO[12] is a new symbolic computation system being developed at the Institute of Systems Science of the Chinese Academy of Sciences. Capabilities of ELIMINO include manipulation of multi-precision numbers and polynomials, computation of characteristic set in Wu's method[23], polynomial equation solving, geometric theorem proving, etc..

To make ELIMINO easy to use and widely accessible, a good, standardized programming interface or application framework is needed. Preferably, this interface or framework can not only be adapted to a good graphical user interface (GUI) for end users, but also fit in the future computing environment such as Web service framework. That is, the programming interface or application framework should be able to make ELIMINO a potential core mathematical compute engine in Web service framework.

By implementing OMEI as ELIMINO's application programming interface, we could make ELIMINO engine IAMC compliant and thus achieve an open computation environment that could provide computational services either locally, or remotely over Internet/Web environment, or in Web Service framework. The services will be accessible not only through the graphical user interface based IAMC client, Dragonfly for example, but also by other OMEI compliant tools, such as scientific editors and word processing systems, with minimal efforts.

OMEI Implementation in ELIMINO

There are basically two approaches to the implementation of OMEI for ELIMINO compute engine. The most straightforward approach is to use inter-process communication(IPC) to access ELIMINO. ELIMINO reads from *standard input* and writes to *standard output*. The OMEI driver connects to ELIMINO with a two-way pipe, sending computational requests into the pipe and reading computational results from the pipe. The IPC approach is simple and straightforward, but lacks flexibility and user control.

As current ELIMINO comes with an API of a set of ad-hoc C/C++ functions, we have adopted the same-process approach to implement OMEI for ELIMINO. This approach is more flexible and allows users to set up the computation environment, input/output mode, command encoding format, and resource allocation. In-process data transfer is also more reliable than IPC.

In our same-process approach, the OMEI is implemented above the application

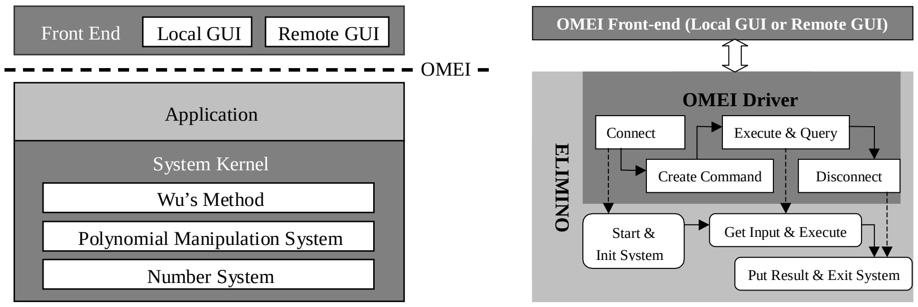


Figure 3: OMEI via Function Calls

layer of ELIMINO, but below the front-end layer (see Figure 3(left)), providing standard programming interface for front-end layer to access computation powers of the ELIMINO compute engine.

Our implementation of OMEI drivers for ELIMINO is a standard C/C++ library. It contains a total of 18 functions for connecting to/disconnecting from ELIMINO, querying ELIMINO's capabilities, creating computational commands and executing these commands, each of them is programmed as the function prototypes specified in the OMEI specification. Figure 3(right)) shows how the OMEI drivers are implemented by function calls in ELIMINO.

The following example shows the OMEI calling pattern to perform the simple computation *gcd((x+y+z)*(7*x-8*z),(7*x-8*z)*(37*z-87*y*x))* by using ELIMINO.

```
   #include <omei.h>
   #include "ELIMINO.h"
   main()
   {  OMEICONHandle *mc;  /*allocate connection handle*/
      OMEICMDHandle *cmd; /*allocate command handle    */
      OMEIString userName = "elim", passwd ="elim";
      OMEIString engineName = "ELIMINO";
      OMEIString mathMLResult;
      OMEIAllocCONHandle(mc);
      OMEIConnect(mc, engineName, userName, passwd);
      OMEIAllocCMDHandle(cmd);
      OMEINativeCommand(cmd, \
                    "gcd((x+y)*(7*x-z),(7*x-z));");
      OMEIResultEncoding(mc, cmd, MATHML);

      OMEIExecuteCommand(mc, cmd);

      OMEIGetResult(mc, cmd, mathMLResult);
```

```
13)   OMEIDisconnect(mc);

14)   OMEIFreeCMDHandle(cmd);

15)   OMEIFreeCONHandle(mc);
   }
```

First, a connection to the compute engine ELIMINO is established (lines 6-7). Then, an OMEI command is created from a literal string (lines 8-9). After specifying the intended encoding for the computation result, the created command is submitted to the connection for execution (line 10-11). The application then retrieves the result (line 12). Finally we close the connection and free the allocated resources (lines 13-15).

6 Applications: Accessing ELIMINO/OMEI

There are basically two approaches to access our computing environment built over OMEI and ELIMINO. As OMEI has been adopted in IAMC framework as the unified interface between IAMC server and compute engine, accessing ELIMINO/OMEI through an IAMC client would be a natural accessing approach For end users, any generic IAMC client software can be used as an interactive utility to access the computational services from ELIMINO engine. For application developers, the MCP protocol library can be intergrated into any application to send MCP requests to an IAMC server to obtain computational services programmatically.

As the OMEI interface is XML compliant, the OMEI/ELIMINO intergrated environment can also be used as the service provider for any XML compliant application, programmatically or interactively. Particularly, ELIMINO/OMEI can be used to deploy a Web service engine. The following we will discuss these two approaches in details.

6.1 Accessing ELIMINO/OMEI in IAMC Framework

Dragonfly is an IAMC client prototype and *Starfish* an IAMC server prototype which are implemented in Java. The Dragonfly-Starfish approach makes interactive mathematical computations accessible and interoperable over the Internet. By adopting OMEI as an application interface, ELIMINO can easily use *Dragonfly* as its local interface (see Figure 4 (a)) or remote user interface (see Figure 4 (b)).

A graphical user interface like Dragonfly not only helps beginning users to get started in ELIMINO quickly, but also provides potentials for adding into ELIMINO more functionalities, such as mathematical graphing and plotting. To interface Dragonfly to ELIMINO as a local GUI, a Java OMEI driver has to be developed because Dragonfly is developed in Java. In other words, Dragonfly has to load this Java OMEI driver to interact with ELIMINO. The Java OMEI driver could be thought of as a Java Wrapper for OMEI function calls and it is implemented by using Java Native Interface (JNI). Figure 5 shows the user interface of Dragonfly as the local GUI for ELIMINO.

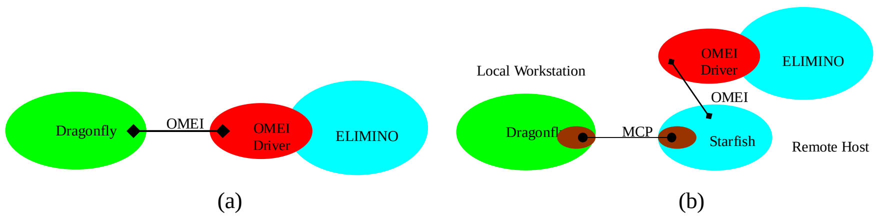


Figure 4: Use Dragonfly as local interface (a) or remote user interface (b)

Using Dragonfly through IAMC as a remote GUI is another story. It is possible to turn the ELIMINO compute engine into an IAMC server by integrating MCP library into it. Nevertheless, with the OMEI interface, it is much easier to integrate ELIMINO into an existing IAMC server, such as our IAMC server prototype *Starfish*. *Starfish* is configured to dynamically locate and load the OMEI driver. With the loaded driver, *Starfish* will be able to forward the computational requests to ELIMINO and results to *Dragonfly*, or any other IAMC clients. Figure 6 shows the user interface of *Dragonfly* as the remote GUI for ELIMINO.

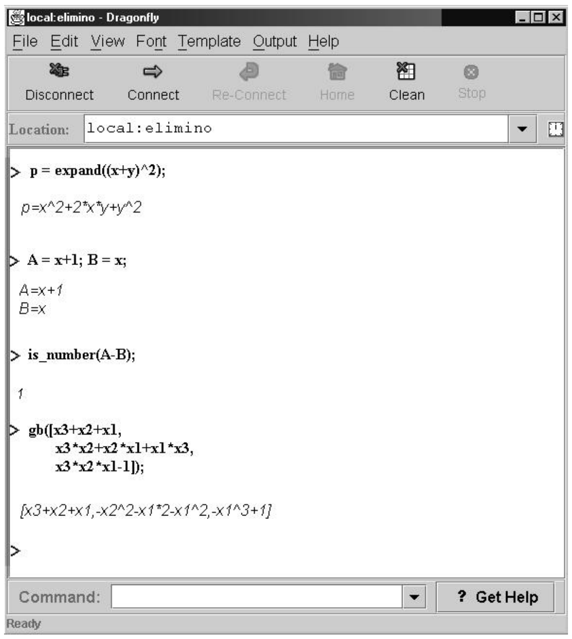
Turning ELIMINO into an IAMC server directly (by integrating MCP library) or indirectly (by using an IAMC server like *Starfish*) also has some other potentials. This will make ELIMINO an IAMC compliant engine and it will be able to serve any MCP requests from Internet. For example, a Web-based mathematical education system may send MCP requests to ELIMINO to get answers for some algebra questions, and a distributed crypt algorithm may issue MCP requests to ELIMINO to get some big prime numbers. In general, Java developers now can use the Java based MCP library [20] to send computational requests to an IAMC server such as Starfish to obtain computational services programmatically.

6.2 OMEI/ELIMINO as a Web Service Provider

The World Wide Web is more and more used for application to application communication. The programmatical interface made available are referred to as *Web services* [21]. The Web service model has been rapidly adopted to build distributed applications to date. Due to its compliance with XML/MathML, OMEI can naturally fit into Web service framework, and OMEI/ELIMINO interface makes it easy and straightforward to develop mathematics oriented Web services based on ELIMINO engine.

Before we discuss the Web service scenario based on OMEI/ELIMINO, let us take a look at how Web service works. Web service framework consists of a group of protocols and specification. The following are essential for Web services to work:

- *Web Services* expose useful functionality to Web users through a standard Web protocol. In most cases, the protocol used is SOAP.
- *Web services* provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them. This description is usually



```
> p = expand((x+y)^2);

 p=x^2+2*x*y+y^2

> A = x+1; B = x;

 A=x+1
 B=x

> is_number(A-B);

 1

> gb([x3+x2+x1,
      x3*x2+x2*x1+x1*x3,
      x3*x2*x1-1]);

 [x3+x2+x1,-x2^2-x1*2-x1^2,-x1^3+1]

>
```

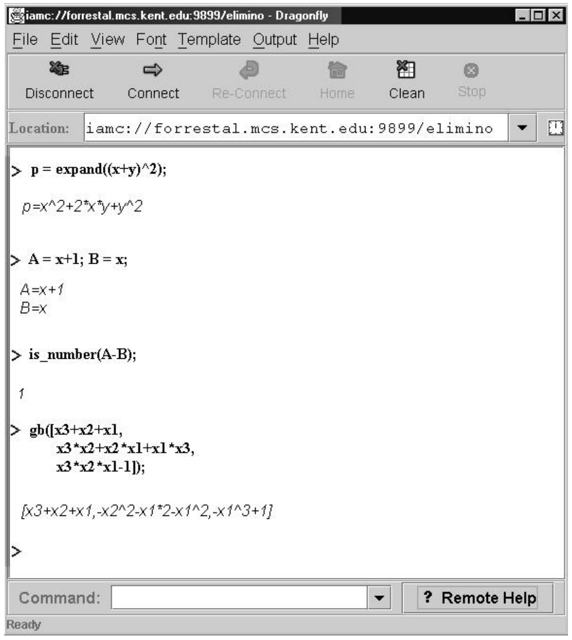
Figure 5: Dragonfly as a local GUI

provided in an XML document called a Web Services Description Language (WSDL) [22] document.

- *Web services* are registered so that potential users can find them easily. This is done with Universal Discovery Description and Integration (UDDI) [16].

Web services depend on the ability of components to communicate with each other even if they are heterogeneous systems. The common encoding mechanism is crucial to the success of Web services and the encoding mechanism adopted by Web services is XML. The implementation of ELIMINO/OMEI has inherent support for XML/MathML encoding and thus can be used to build and deploy Web services easily and quickly.

The XML/MathML support for ELIMINO/OMEI is built over XMEC [24] package, an eXtensible Mathematical Encoding Converter into Java OMEI driver for ELIMINO engine so that the engine could accept input and produce output encoding in standard encoding formats, such as MathML. The ELIMINO/OMEI environment itself now can accept and evaluate XML/MathML code and generate MathML output. It provides sematics and document oriented processing capabilities for XML/MathML. Other procedure oriented XML processing, such as SOAP message passing and Registries, will still rely on generic development environment. Currently, both J2EE and Visual Studio .Net provide generic APIs and frameworks for Web serices development.


iamc://forrestal.mcs.kent.edu:9899/elimino - Dragonfly
File Edit View Font Template Output Help
Disconnect
Connect
Re-Connect
Home
Clean
Stop
Location:
Command:
? Remote Help
Ready

```
iamc://forrestal.mcs.kent.edu:9899/elimino

> p = expand((x+y)^2);

  p=x^2+2*x*y+y^2

> A = x+1; B = x;

  A=x+1
  B=x

> is_number(A-B);

  1

> gb([x3+x2+x1,
      x3*x2+x2*x1+x1*x3,
      x3*x2*x1-1]);

  [x3+x2+x1,-x2^2-x1*2-x1^2,-x1^3+1]

>
```

Figure 6: Dragonfly as a remote GUI

7 Conclusion

The OMEI effort investigates ways to establish a uniform API for mathematical compute engines and how such a uniform API can make distributed mathematical computation easier. We have applied the OMEI specification in ELIMINO, new computer-mathematics research system developed in China. The OMEI interface entitles several new features for ELIMINO, a new computer-mathematics research system, with minimal efforts. An existing graphical user interface, *Dragonfly*, could be used to access ELIMINO locally and remotely. As an IAMC compliant compute engine, ELIMINO can be used as a compute server for Internet based mathematical computational and education systems. Moreover, OMEI also help ELIMINO gain interoperability with other OMEI compliant tools, such as scientific editors and word processing systems.

Acknowledgements

Work reported herein has been supported in part by the USA National Science Foundation under Grant CCR-9721343 and INT-9722919, and the Ohio Board of Regents Computer Science Enhancement Funds, the Chinese National 973 Project NKBRSF G1998030609, Chinese 863 Project 2001AA144030.

References

[1] ABBOTT, J., DIAZ., A., AND SUTOR, R. S. *Report on OpenMath.* ACM SIGSAM Bulletin (Mar. 1996), 21-24.

[2] DOLEH, Y., AND WANG, P. S. *SUI: A System Independent User Interface for an Integrated Scientific Computing Environment.* In Proc. ISSAC 90 (Aug. 1990), Addison-Wesley (ISBN 0-201-54892-5), pp. 88-95.

[3] GRAY, S., KAJLER, N., AND WANG, P. *Design and Implementation of MP, A Protocol for Efficient Exchange of Mathematical Expressions.* Journal of Symbolic Computation 25 (Feb. 1998), 213-238.

[4] ION, P., MINER, R., BUSWELL, S., S. DEVITT, A. D., POPPELIER, N., SMITH, B., SOIFFER, N., SUTOR, R., AND WATT,S. *Mathematical Markup Language (MathML) 1.0 Specification.* (www.w3.org/TR/1998/REC-MathML-19980407), Apr. 1998.

[5] JavaMath. *http://javamath.sourceforge.net/.*

[6] KAJLER, N. *CAS/PI: a Portable and Extensible Interface for Computer Algebra Systems.* Proceedings of ISSAC'92, ACM Press 1992

[7] KQML Home Page. http://www.csee.umbc.edu/kqml.

[8] LIAO, W. and WANG, P. S. *Building IAMC: A Layered Approach.* Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'00). pp. 1509-1516.

[9] LIAO, W. and WANG, P. S. *Dragonfly: A Java-based IAMC Client Prototype.* Lecture Notes on Computing Vol.8. (Proceedings of ASCM 2000.) World Scientific Press, pp. 281-290.

[10] LIAO, W. and WANG, P. S. *Specification of OMEI: Open Mathematical Engine Interface.* ICM Technical Report. 2001. http://icm.mcs.kent.edu/reports/index.html.

[11] LIAO W., LIN D. and WANG P. S. *OMEI: Open Mathematical Engine Interface.* Proceedings of ASCM'2001, pp 83-91, Matsuyama, Japan, September 26-28, 2001. Lecture Notes Series on Computing Vol. 9, World Scientific.

[12] LIN D., LIU J. and LIU Z. *Mathematical Research Software: ELIMINO.* Proceedings of ASCM'98. pp. 107 - 116, Lanzhou Univ., China, 1998.

[13] MathLink Home Page. http://www.wolfram.com/solutions/mathlink.

[14] MATLAB External Interface. http://www.mathworks.com/access/helpdesk/help/techdoc/ matlab.shtml.

[15] Simple Object Access Protocol (SOAP) 1.1. W3C Note. 08 May 2000. http://www.w3.org/TR/SOAP/

[16] Universal Description Discovery and Integration (UDDI) Version 3.0. Published Specification, 19 July 2002. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm

[17] WANG, P. S. *Internet Accessible Mathematical Computation.* In the 3rd Asian Symp. on Computer Mathematics (ASCM'98), pp 1-13, Lanzhou Univ., China, 1998.

[18] WANG, P. S., GRAY, S., KAJLER N., LIN D., LIAO W. etc. *IAMC Architecture and Prototyping: A Progress Report.* Proceedings of ACM ISSAC'01, University of Western Ontario, London, Ontario, Canada, July 22-25, 2001.

[19] WANG, P. S. *Design and Protocol for Internet Accessible Mathematical Computation.* In Proc. ISSAC'99 (1999), ACM Press, pp. 291-298.

[20] WANG, P. S., GUO, Q. and LIAO, W. *Mathematics over the Internet/Web: A Protocol-based Approach.* ICM Report No. ICM-200206-0007. http://icm.mcs.kent.edu/reports/index.html.

[21] Web Service Activity inside W3C. http://www.w3.org/2002/ws/.

[22] Web Services Description Language (WSDL) 1.1. W3C Note. 15 March 2001. http://www.w3.org/TR/wsdl.

[23] WU, W. T. *Basic Principle of Mechanical Theorem Proving in Elementary Geometries,* J. Syst. Sci. Math. Sci. 4, 207-235 (1984).

[24] ZOU, X. *XMEC: An Extensible Mathematical Encoding Converter.* ICM Technical Report. 2001. http://icm.mcs.kent.edu/reports/index.html.

[25] YOUNG, D. and WANG, P. S. *GI/S: A Graphical User Interface for Symbolic Computation Systems.* In Journal of Symbolic Computation. Vol 4. No. 3. 1987. pp. 365-380.