Wei Wu, M.S., May, 1998 COMPUTER SCIENCE

EXPERIMENTS WITH INTERNET ACCESSIBLE MATHEMATICAL COMPUTATION (55 pp.)

Director of Thesis: Dr. Paul S. Wang

The goal of Internet Accessible Mathematical Computation (IAMC) research is to design a system to let the user access remote mathematical computation systems based on the Internet through any computer platform machine, and to let the user do numeric, symbolic, and other scientific computations. Internet users can input their mathematical request, send it to the remote mathematical computation server, and get the answer back.

The work here constitutes a simple proof-of-concept system where a simple IAMC client-server pair is written in Java, MP is used for mathematical data encoding and transfer, and the pair provides Internet-based access to the MAXIMA symbolic computation system. By this experiment, we hope to shed more light on the IAMC approach and to provide experience for a full-blown IAMC prototype.

EXPERIMENTS WITH INTERNET ACCESSIBLE MATHEMATICAL COMPUTATION

A thesis submitted
to Kent State University in
partial fulfillment of the requirements
for the degree of Master of Science

by

Wei Wu

May, 1998

Thesis written by

Wei Wu

M.S., Kent State University, 1998

Approved by

______________________________, Advisor

______________________________, Chair, Department of Mathematics and Computer Science

______________________________, Dean, College of Arts and Sciences

TABLE OF CONTENTS

LIST OF FIGURES v

LIST OF TABLES vi

Acknowledgements vii

1 Introduction 1

1.1 Motivation 1

1.2 Previous Work 2

1.3 IAMC 7

2 Design of IAMC 10

2.1 Overview of IAMC System 10

2.2 IAMC Client 14

2.3 IAMC Server 16

2.4 Object-Oriented Design 17

3 IAMC Client 21

3.1 Parsing 21

3.1.1 Infix Input 21

3.1.2 Recursive Descent Parsing 23

3.1.3 IAMC Math-Expression Grammar 26

3.1.4 Tokens 28

3.1.5 Internal Representation 28

3.1.6 Prefix Notation Generation 29

3.2 ASCII–MP Conversions 32

3.2.1 Prefix to MP Conversion 32

3.2.2 MP to ASCII Conversions 33

3.3 Interface to IAMC Server 33

4 IAMC Server 35

4.1 Interface to IAMC Client 35

4.2 The Compute Engine 36

4.3 Interface to Compute Engine 37

4.4 Server-Client Communication 40

5 IAMC Usage Examples 44

5.1 Example 1 44

5.2 Example 2 45

5.3 Example 3 46

5.4 Example 4 47

6 Conclusion 51

BIBLIOGRAPHY 53

LIST OF FIGURES

1 Structure of IAMC 11

2 Programming Contributions 12

3 Flow of Computation Request 19

4 IAMC Classes 20

5 Example of Prefix Generating 31

6 Java Exec Method 32

7 Interface to Compute Engine 43

LIST OF TABLES

1 IAMC Supported Operators 22

Acknowledgements

I would especially like to thank my advisor, Dr. Paul Wang, for his direction, advice, and guidance on this research. I would also like to thank Dr. Robert Walker and Dr. Hassan Peyravi for their suggestion and comments on this project. Also I would like to thank Benjamin Norman for his help on editing my thesis.

CHAPTER 1

Introduction

1.1 Motivation

The Internet [9] is the network that is changing the way people communicate, interact, and define community. The Internet has a spirit of invention within the human community it encompasses as expressed in the tools, software, and hardware that are used. The Internet represents a changing paradigm for human interaction. It is a collaborative medium in which you can access information and data; it is a place for learning, commerce, entertainment, and intensive interaction with people.

A major purpose of the Internet is the sharing of information between universities, companies, governments, nonprofit organizations, and individuals. At present, millions of pieces of information are available on the Internet in a variety of formats, but sharing of mathematical computation and data is lagging behind.

Some universities and organizations have mathematical computation systems, experimental and commercial, which are powerful tools for scientists, engineers, and educators. They aim to automate mathematical computations of all sorts. These tools may be general purpose such as Maple [21], MAXIMA [12], and Mathematica [32] for computer algebra or specialized such as Camal [39] for general relativity and Singular [40] for commutative algebra and algebraic geometry. On the other hand, almost all the powerful computation systems are only accessible directly on their host computers. Usually, outside users can not use these systems remotely without obtaining accounts on these machines and logging in first. It is desirable to make powerful

mathematical and scientific computing systems accessible on the Internet as simply and easily as a Web page.

Some simple tools, such as interactive calculators [38], exist on the Internet for limited mathematical calculation, but a systematic approach for access, control, expression encoding, and interoperability must be devised if complex mathematical-oriented computation services are to become usable on the Internet via the client-server paradigm. Some existing systems such as SUI [13] can let the user connect to multiple powerful compute engines, but they have the problem of platform limitation: they only can be used on Unix machine.

The goal of Internet Accessible Mathematical Computation (IAMC) research is to design a system to let the user access remote mathematical computation systems based on the Internet through any computer platform machine, and to let the user do numeric, symbolic, and other scientific computations. Internet users can input their mathematical request, send it to the remote mathematical computation server, and get the answer back. For example: a user in China wants to know the factors of polynomial *x^10-1*. He/she can use his/her home PC to access the Internet by any browser and run the IAMC Client service in Kent. On the IAMC Client interface, the user can input the compute request and select the IAMC Server in New York which has a Maple compute engine. After the user inputs his/her request, the request will be sent to New York. The answer will come back from New York and will be displayed on the China user's home PC screen.

1.2 Previous Work

Sharing mathematical computation and data is not a new problem for scientific computing. There are different sub-areas for this problem.

Firstly, sharing mathematical computation and data requires a mathematical user interface which handles the user input, parses the input, and acts as a client to the computation server. A well-designed user interface not only makes a computing system more appealing and convenient to use but also increases the computation power provided to the user.

Different aspects of the user-interface problem have been successfully addressed by different realizations. Two-dimensional editing of mathematical formulas is achieved in experimental systems such as MathScribe [17], GI/S [5], or SUI [1] and in recent mathematical assistants such as Milo [19] and Theorist [34]; some kinds of hypertextual access to the documentation is provided by the graphical user interfaces of both Mathematica and Axiom [22]. Also, quality curve and surface plotting is performed by Axiom, Maple V, Mathematica, and SUI.

An important objective of improving these components is to have all these facilities gathered in a single software environment and to make such an environment independent of a given product or system. Also, few user interfaces support concurrent use of different computer algebra systems, faced with conversion problems, configuration management, and communication protocols. Pioneering work on these improvements was done by James Purtilo with Polyith [20] and Denis Arnon with CaminoReal [18]. More recent results include MathStation [25], SUI, and CAS/PI [2].

Secondly, mathematical computation needs an efficient and standard encoding format for data exchange and communication. Traditionally, computer algebra systems are monolithic, stand-alone programs designed to communicate with a user through a specific command language. They do not address interoperability issues such as

the exchange of mathematical expressions with other independent programs. More recently, scientific computation systems have adopted the component approach and devised various schemes for inter-component communication.

Since version V, Maple has been composed of a kernel and a set of devices including a user interface, Iris [24], and a plotting engine. The kernel and devices can run on different computers communicating with a proprietary protocol. Data can be passed in one of two ways: either as string suitable to be used as Maple input, or as Directed Acyclic Graphs (DAGs) using Maple's internal data representations. DAGs have two clear advantages: first, they reduce the average amount of data transmitted by sharing common subexpressions; second, using Maple's internal data representation eases data encoding and decoding on the kernel side. Recently, Maple introduced MathEdge [27], a development toolkit which enables developers to link their applications with the Maple kernel.

Beginning with version 2, Mathematica communicates using MathLink [33]. MathLink implements a communication protocol and provides a set of procedural interfaces that allow C programs to send and receive data or call (or be called by) Mathematica, or allow different instances of Mathematica to communicate with each other. MathLink is fully documented and library routines are provided for advanced users to write their own applications.

The POSSO [31] project is one of the European research projects centered on symbolic computation. It includes the definition of two protocols for exchanging mathematical expressions, XDR-POSSO [31] and ASAP [30]. XDR-POSSO is designed to exchange POSSO data structures using a binary encoding based on the XDR technology. It is strongly tied to the POSSO project and does not include

annotations or a general extension mechanism to support other kinds of mathematical objects. ASAP (A Simple ASCII Protocol) is more oriented towards portable exchange of mathematical expressions encoded as linearized attributed trees. It provides a basic technology, relying on the user to define the semantics of the expressions exchanged and to provide more optimized encodings when appropriate.

Euromath [28], another EEC-funded project, defines a Document Type Definition (the Euromath DTD) for SGML [36], which formally specifies the structure of the mathematical objects to be exchanged and provides the GRIF [28] editor for editing them.

Mathematical Markup Language (MathML) [15] is the official HTML[37] extension sponsored by the W3 (World Wide Web) consortium. MathML is an XML [35] application for describing mathematical notation and capturing both its structure and content. The goal of MathML is to enable mathematics to be served, received, and processed on the Web, just as HTML has enabled this functionality for text. MathML also pays particular attention to compatibility with other mathematical software, and in particular, with computer algebra systems. Many of the presentation elements of MathML are derived in part from the mechanism of typesetting boxes. The MathML content elements are heavily indebted to the OpenMath [26] project. The OpenMath project has close ties to both the SGML [36] and computer algebra communities, and has laid a foundation for an SGML-based means of communication between mathematical software packages, among other things. The feasibility of both generating and interpreting MathML in computer algebra systems has been demonstrated by prototype software.

Several other works related to the exchange of mathematical data between scientific applications are reported in [4]. Notable implementations of distributed architectures that provide some exchange of mathematical expressions include Polylith [20], CaminoReal [18], SUI, DSC [29], and CAS/PI. Realizing the importance of a mathematical data exchange protocol, the Maple group initialized a series of workshops which led to the formation of the OpenMath group in order to develop and standardize such a protocol.

Multi Protocol (MP) [3] is a result of collaboration among Gray, Kajler, and Wang. MP is a format for efficient communication of mathematical data among scientific computing systems. MP is designed to support efficient communication of mathematical data between scientifically-oriented software tools. MP exchanges data in the form of linearized annotated syntax trees. Syntax trees provide a simple, flexible and tool-independent way to represent and exchange data, and annotations provide a powerful and generic expressive facility for transmitting additional information. In a layer above the data exchange portion of the protocol, MP supports collections of definitions for annotations and mathematical symbols (operators and symbolic constants) in dictionaries. Dictionaries address the problem of application heterogeneity by supplying a standardized representation and semantics for mathematical objects. They are identified within packets through a dictionary tag field. Applications that communicate according to definitions provided in dictionaries do not need to have direct knowledge of each other, promoting a "plug-and-play" style of interoperation of mathematical expressions at the application level.

1.3 IAMC

Wang's research group at the Institute for Computation Mathematics at Kent State University and collaborators, including Prof. Dieter Schmidt (Electrical and Computer Engineering, Cincinnati), Prof. Simon Gray (Mathematics and Computer Science, Ashland), and Prof. Norbert Kajler (Ecole des Mines de Paris, France), are involved with the conceptualization, architecture, design, and prototyping of *Internet Accessible Mathematical Computation* (IAMC). The goal, simply put, is that on the Internet a user should be able to access mathematical computation services directly. It should be as simple and easy as accessing a Web page or sending email.

The IAMC research has these goals:

- To make math-oriented data and services easily accessible on the Internet – directly, via the Web, and by email.
- To allow remote *compute servers* usable almost like local programs.
- To enable effective and efficient communication of mathematical data over the Internet.
- To make it possible for heterogeneous compute servers to exchange computational results and perform further computation on them.

The evolving IAMC architecture calls for an IAMC client to provide a GUI front end, an IAMC server to provide or connect to a computation service, and a *Mathematical Computation Protocol* (MCP) to connect the client and server.

IAMC applications may include:

- Use of remote computation services.

- Access to remote scientific databases.
- Making parallel/super computing accessible.
- Distance learning.
- Computing via NetPC for high school or occasional users.
- Establishing *Problem Solving Environments* (PSE).

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

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

IAMC services should be available via TCP/IP [8], the Web, or email.

The work reported here constitutes a simple proof-of-concept system where a simple IAMC client-server pair is written in Java, MP is used for mathematical data encoding and transfer, and the pair provides Internet-based access to the MAXIMA symbolic computation system. By this experiment, we hope to shed more light on the

IAMC approach and to provide experience for a full-blown IAMC prototype. From this point on, the name IAMC refers to the proof-of-concept system.

CHAPTER 2

Design of IAMC

The current IAMC system is a bare-bones version of an IAMC system and builds an experimental client-server package. This simple proof-of-concept system uses MP for mathematical data encoding and transfer, implements IAMC client-server pair in Java, and provides Internet-based access to the MAXIMA symbolic computation system.

2.1 Overview of IAMC System

The IAMC system has two parts: one is IAMC Client, the other is IAMC Server. Figure 1 shows the relations of the two parts connected by stream sockets [7].

The programming of the work reported here includes the Java-coded client and server parts, i.e., all the classes and functions in the IAMC Client and Server, as well as the ANSI C coded MP conversion programs `prefix2mp` and `maxima2mp`. The `prefix2mp` program is used for converting prefix notation to MP format; the `maxima2mp` program is used for converting MAXIMA output to MP format. The `displaymp` program is written by Paul Wang and it is used to convert MP data to different formats such as infix, prefix, and LaTeX. MP library is written by Simon Gray as part of his MP project. Figure 2 shows the overall programming contributions.

IAMC Client will be accessed by the user, accept the user input and display the output to the user. IAMC Server will connect to the local compute engine and control the compute engine to do the mathematical computation. IAMC uses the

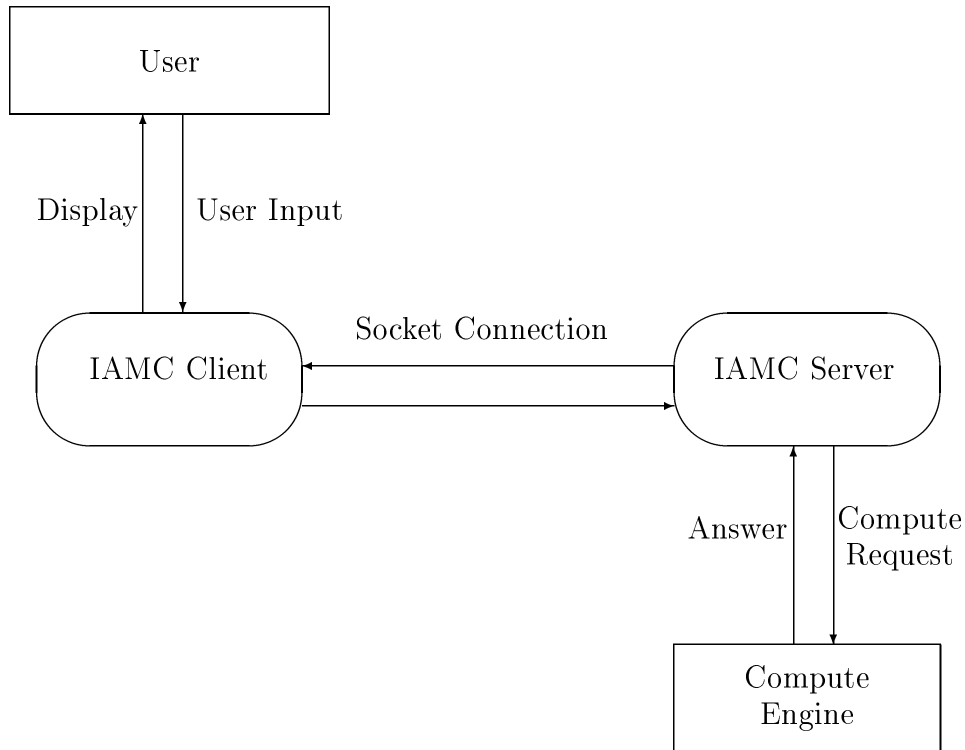

Figure 1: Structure of IAMC

Client/Server model for IAMC Client and IAMC Server communication.

Usually a network application will involve a server and a client. A server process provides a specific service on a host machine that offers such a service. Example services are remote host access (telnet), file transfer (ftp), and the World Wide Web (http). Each network-wide service has its own unique port number that is identical across all hosts. The port number together with the Internet address of a host identifies a particular server anywhere on the Network. For example, *ftp* has port number 21, and *http* 80. Currently, the first 512 port numbers (0-511) are reserved for network-wide applications registered by the InterNIC. The next 512 ports (512-1023)

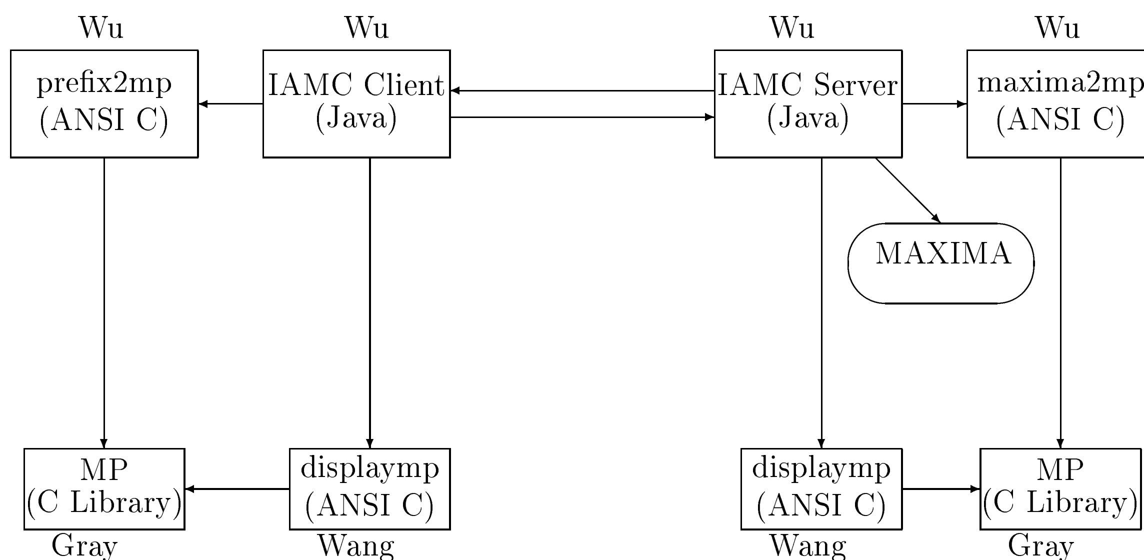

Figure 2: Programming Contributions

are semi-official and are used for such standard services as remote UNIX login at 513 and remote printing at 515. Still higher port numbers are used for local applications such as X Windows and NFS [8]. IAMC uses port number 4450 for experiments.

A client process on a host connects with a server on another host to obtain its service. Thus, a client program is the agent through which a particular network service can be obtained. Different agents are usually required for different services.

The IAMC Client sits at the user's site while IAMC Server locates on the compute engine machine. The IAMC Server should have the right to access the compute engine, because it needs to invoke the mathematical compute engine and do the user requested job. From the user's point of view, the IAMC server is located on a remote machine which supplies such computation service. The user doesn't need to know where it is; he/she just needs to know the IAMC Server's name (like `ox.mcs.kent.edu`) and the port number which is used for IAMC computation. Hopefully all the IAMC

Servers have unique port number to do such computation, just like 80 for http and 70 for gopher. In that case the user just inputs the server's name and the unique port number will be automatically assigned to the system. IAMC uses MP as the data communication format between the IAMC Client and IAMC Server. IAMC Client will translate all requests into MP format and send to the IAMC server. For IAMC Server's part, it should understand MP, and convert the MP data to the appropriate data which is the input of the actual compute engine, after the compute engine finishes the requested job, the IAMC Server should collect the answer from the compute engine and translate to MP data, then send the result back to the client.

IAMC Client and Server are implemented in Java [10] [7]. The Java programming system is one of the most exciting recent developments in computing. Developed by Sun Micro Systems, Java gained world-wide acceptance with unprecedented speed. It is primarily due to Java's simple yet powerful mechanisms for object-oriented programming (OOP) and network-based computing, two dominating trends shaping the future of the computer and information industries. IAMC system chose Java as the implementing language because IAMC is based on the Internet and Java is the best networking language so far. The team that designed Java was well aware that if Java was to support applications on networks, it would have to support a variety of systems with a variety of CPU and operating system architectures. A Java application can execute anywhere on the network, or on the Internet, because the compiler generates an architecture-neutral object file. Once a Java program is written, it stays written. It doesn't need to be recompiled for each different platform because Java programs are compiled into Java bytecodes that run on any computer with a Java interpreter. The bytecodes are architecture and operating system independent. Thus, Java programs

can be written and compiled once and then transmitted to run anywhere. Another important reason for IAMC to choose Java is that since Java is an OOP language, it can be extended very easily. We can add more operators and functions for the mathematical computation.

2.2 IAMC Client

The IAMC client acts as an interface to the user. The IAMC client will read the user's input and display the output to the user. After the IAMC Client gets the user's Fortran-like infix mathematical expression input, it will perform a grammar check, that is, parse the input and tell the user whether the parse is successful or not. When the user wants to send the request to the remote IAMC Server, IAMC will translate the parsed data to MP format, open a TCP/IP socket to send the MP data to the IAMC Server according to its name and port number, and wait for the answer from the IAMC Server. In general, the result can be mathematical expressions in MP format, help information in HTML, 2D or 3D plots in `.gif`, or other well-defined information formats. The experiment here focuses on results in MP format. The answer returned by the server is then displayed for the user. The IAMC client uses a simple MP to infix converter which displays the result in infix form. This converter can be extended to convert MP to prefix notation, MathML, or even LaTeX. Future work can add a nice GUI front end for the IAMC client. Figure 3 shows the flow of one computation request.

The connection between the IAMC Server and Client is session-based. Once the user opens a new session to an IAMC Server, the IAMC Client is connected to that particular IAMC Server. The user can send as many requests as he/she wants to the IAMC Server. After the user closes the session, the connection between the IAMC

Client and Server is gone; there is no context saved for the server. So after the user opens another session, the IAMC will not remember the previous session's context, such as variable assignments. It is possible for the IAMC client to save a sequence of user input commands in a file which can be *batched* to a server. This feature allows a user to create IAMC command files or to save a session to be continued later. This feature is not currently implemented.

The IAMC Client also has the responsibility to translate between the infix and MP, save the MP formatted data to an `.mp` file for future use, and send an exiting MP file to the server. Another duty for the IAMC Client is remembering the user's input history and redisplaying it in order for the user to easily re-issue the earlier input. Finally, the IAMC Client provides help facilities to help the user understand the supported operators, input format, remote server's information, etc. We can summarize the functions that the experimental IAMC Client supports:

- **Get User Input:** read user's input.
- **Grammar Check:** parse the input.
- **Open MP File:** convert and display MP data.
- **Save MP File:** convert to MP formatted data.
- **New Session:** open the connection to IAMC Server.
- **Send Computation Request:** send one MP file to Server.
- **Close Session:** close the current connection with IAMC Server.
- **History:** display user's input history.

- **Help:** display help information.
- **Exit:** exit IAMC System.

2.3 IAMC Server

The IAMC Server connects to the IAMC client on one end and controls an external compute engine on the other end. The IAMC server has two parts:

- The part that deals with the IAMC client is generic.
- The part that controls the external compute engine is specific to the engine.

The interface to IAMC Client is generic. All the IAMC Servers should have common responsibilities. IAMC Server must have a specific IP address and port number, and should always be listening for a connection from the IAMC Client. IAMC Server should be able to access the actual mathematical compute engine, not do computation itself; otherwise it is just another on-line calculator. Because the data communication between IAMC Client and IAMC Server uses MP, all the IAMC servers should support MP architecture.

The part which controls the external compute engine is specific to the actual engine. IAMC Server acts as an "adapter" for the particular compute engine, so the IAMC Server must be able to convert the MP data to the appropriate format as input for that particular compute engine. For example: for MAXIMA the request for getting the factors of polynomial *x^2-y^2* is the command:

```
factor(x^2-y^2);
```

So an IAMC Server must understand its compute engine's language and know how to start the engine, how to feed the input and in what format, how to get the answers,

and how to terminate the engine. Lastly, after the IAMC Server obtains the answers from the compute engine, it will convert it to MP data and send it back to the Client. Currently, the IAMC Server uses the server `ox.mcs.kent.edu`, which is a UNIX machine in the Department of Computer Science at Kent State University. It connects to the local mathematical compute engine MAXIMA, so now the IAMC Server is specific for MAXIMA.

2.4 Object-Oriented Design

One important advantage of Java is its object orientation. The OO design of the IAMC Client consists of the following classes:

- **IAMCClient:** manages the IAMC Client. This is the main control class. It can read the user input, invoke other objects, and communicate with the IAMC Server.

- **Grammar:** performs the grammar check and generates prefix notation expressions.

- **Operator:** represents a mathematical operator used in parsing and prefix generation.

- **Operand:** represents a mathmatical expression used in parsing and prefix generation.

- **Mp:** performs the conversions between MP and any other format.

- **Prefix:** converts any prefix string (internal representation) to MP format.

- **Menu:** displays the main menu of IAMC Client.

- **Help:** provides all help information.
- **History:** displays the history of user inputs.
- **SendMP:** send the MP data (including ending marker) to server.
- **WriteToFile:** stores the MP data coming from the server to an MP file (excluding the ending marker).
- **ArbCell, ArbList, ArbStack:** are generic container classes from Wang's Java book [7]. They are excellent examples of generic reusable codes. IAMC Client uses the ArbStack class for generating prefix notation.

The IAMC Server is relative simple. It has only two classes:

- **IAMCServer**: serves as the main control class to perform communications with IAMC Client and the actual compute engine.
- **WriteToFile** (reuse): writes the MP data coming from the client to an MP file.

Figure 4 shows the relation of these classes.

With the above-described OO design, it is very easy and clear to extend the existing functions. For example, if one wants to extend the grammar rule, he/she only needs to extend the `Grammar` class. If a new operator has more information, one can simply extend the `Operator` class.

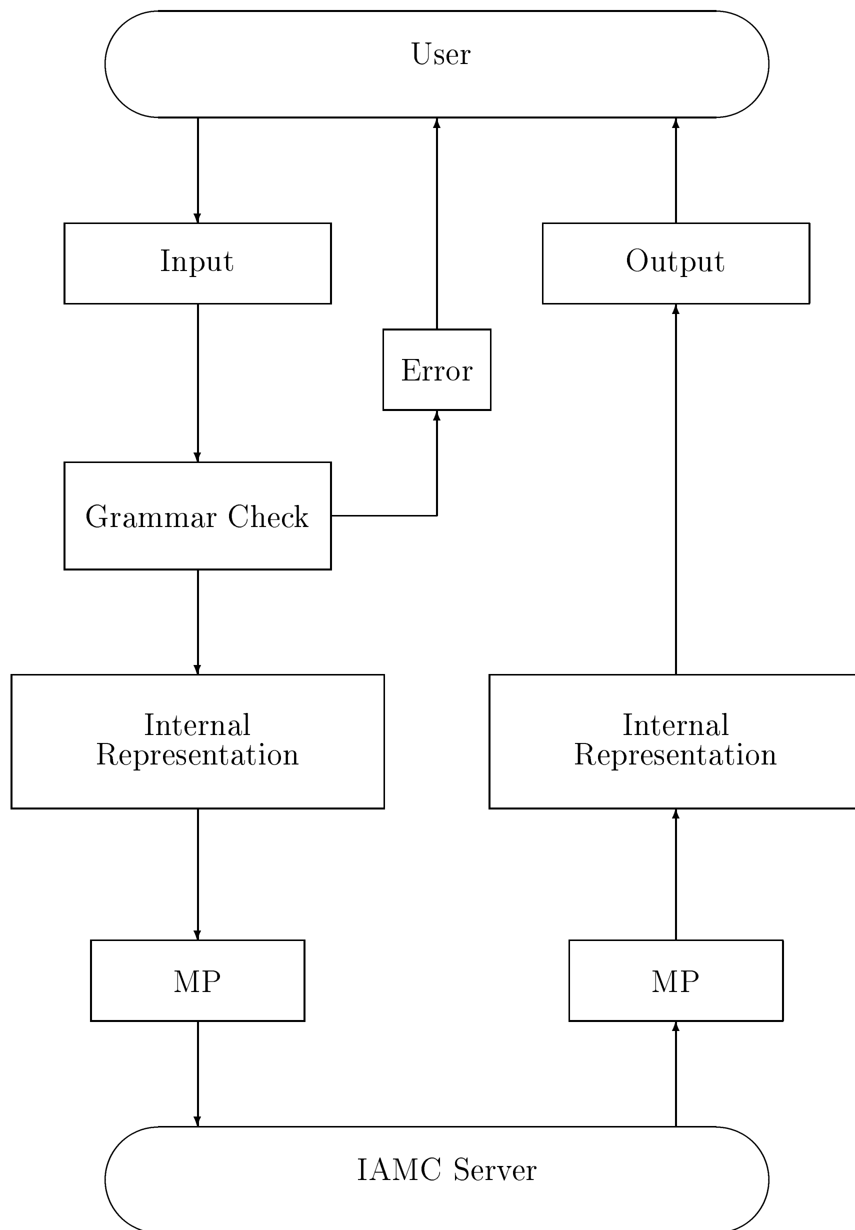

Figure 3: Flow of Computation Request

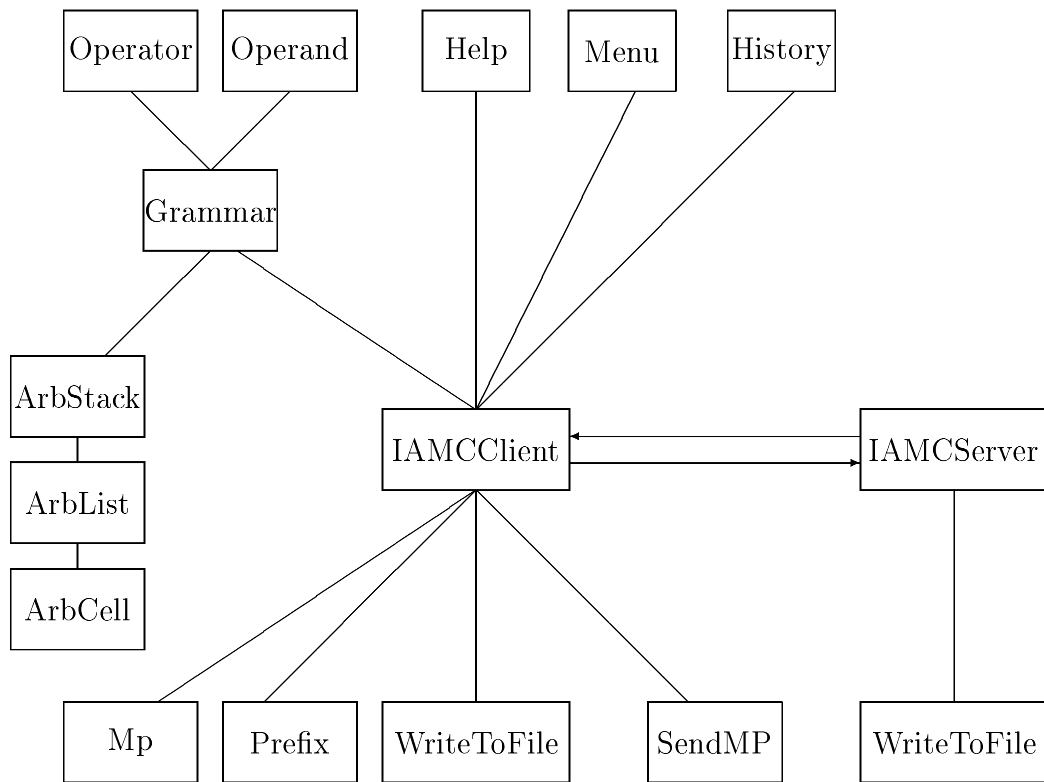

Figure 4: IAMC Classes

CHAPTER 3

IAMC Client

This chapter will show the details of how the IAMC Client part is implemented. It will emphasize the input parsing part; the conversion between ASCII and MP and the interface to IAMC Server also will be described.

3.1 Parsing

The IAMC Client receives mathematical expressions and commands from the user. The input is parsed and converted to MP form by a simple recursive descent parser with a grammar that supports integer and floating numbers, identifiers, special mathematical constants, mathematical functions, and operators.

3.1.1 Infix Input

User input is entered in infix notation similar to Fortran. For example:

```
x*y-(z/x-(y+2))
sin(x*2-y*2)+cos(z)
```

Special constants include such quantities as "pi", "i", and "e". IAMC Client supports the standard operators in Table 1 since they are common operators: their syntax, number and order of arguments are quite standard. The IAMC Client grammar will expect the supported operators to have the exact correct number and order of arguments. All the other operators not listed in Table 1, such as `limit, sum,` and `prod,` are considered unsupported operators. That means their syntax are not

Operator Name	Representation
ADDITION	+
SUBTRACTION	-
MULTIPLICATION	*
DIVISION	/
POWER	^
SINE	sin
COSINE	cos
TANGENT	tan
COTANGENT	cot
ARCSINE	asin
ARCCOSINE	acos
ARCTANGENT	atan
ARCCOTANGENT	acot
SQUARE ROOT	sqrt
EXPONENTIATION	exp
NATURAL LOGARITHM	log
FACTOR	factor
DIFFERENTIATE	diff
INTEGRATE	integrate
EXPAND	expand

Table 1: IAMC Supported Operators

standard; their number and order of arguments are different according to different systems. They also can get parsed successfully as long as they are input as

`operator` (*arg1*, *arg2*, ... *argN*)

where (*arg1*, *arg2*, ... *argN* are math expressions). For example, if one inputs:

```
sum (x^2+x, x, 1, 9)
```

although the `sum` is not a supported operator, it satisfied the rule and it will be correctly parsed (while its syntax is unknown). Even if one inputs:

`xyz` (x)

it will be also parsed successfully and it will be considered that the operator name is `xyz` and it has one argument x.

Variable (Identifier) is any string which begins with letter and followed by letter or digit. Number is any digit combination including one "." for real numbers. Some of the supported operators, such as `sin` and `factor`, have only one argument (operand).

```
sin(x+y)
factor(x^6-1)
```

while some operators, such as `integrate`, have varied arguments (operands). Use the command

`integrate(`*exp, var*`)`

to find the indefinite integral of the expression *exp* with respect to the variable *var*. On the other hand,

`integrate(`*exp, var, high, low*`)`

will be used to find the definite integral of the expression *exp* with respect to the variable *var* from *low* to *high*.

3.1.2 Recursive Descent Parsing

For a given grammar, there are different parsing technologies to choose from. Here, the IAMC Client uses *Recursive Descent Parsing* [14] because it is simple, effective, and easy to extend. One approach to writing a recursive descent parser is:

All legal strings in the language must be derived from the start nonterminal `S`. Hence if we can write a procedure `S()` which matches all strings derivable from `S`,

then we are done. In order to do this, for each nonterminal `N`, we will construct a parsing procedure `N()` which matches all strings derivable from `N`.

If a nonterminal `N` has only a single grammar rule, then its parsing procedure is easy to write. To match the rule we know that we need to match each grammar symbol on its right-hand side (RHS). There are two possibilities:

1. If the RHS symbol is a terminal symbol, check that the current lookahead matches that terminal symbol. If it does, then advance the input, setting the current lookahead to the next input symbol. If it does not, signal an error.

2. For each occurrence of a nonterminal symbol on the RHS, call, in sequential order, the corresponding parsing procedure. A match will consume a prefix of the input with a rule for the nonterminal.

If a nonterminal `N` has multiple grammar rules, then the parsing procedure will need to decide which rule to use. It can do so by looking at the current lookahead token to see which of the candidate rules can start with the lookahead. If only a single rule can start with the lookahead, then that rule is chosen. If it is always possible to predict a unique rule for any nonterminal by looking ahead by at most one token, then the grammar is said to be LL(1).

Hence a parsing function `N()` for nonterminal `N` will simply contain logic to pick a rule based on the current lookahead, followed by code to match each individual rule. Its structure will look something like the following:

```
N()
{
        if (lookahead can start first rule for N)
        {
                match rule 1 for N
        }
        else if (lookahead can start second rule for N)
        {
```

```
                match rule 2 for N
        }
        ...             ...
        else if (lookahead can start n'th rule for N)
        {
                match rule n for N
        }
        else
        {
                error();
        }
}
```

Unfortunately, there are some problems with this simple scheme. A grammar rule is said to be directly left recursive if the first symbol on the RHS is identical to the LHS nonterminal. Hence, after a directly left recursive rule has been selected, the first action of the corresponding parsing procedure will be to call itself immediately, without consuming any of the input. It should be clear that such a recursive call will never terminate. Hence a recursive descent parser cannot be written for a grammar which contains such directly (or indirectly) left recursive rules; in fact, the grammar cannot be LL(1) in the presence of such rules.

Fortunately, it is possible to transform the grammar to remove left recursive rules. Consider the left recursive nonterminal `A` defined by the following rules:

```
A: A alpha
A: beta
```

where `alpha` is nonempty and `alpha` and `beta` stand for any sequence of terminal and nonterminal grammar symbols. Looking at the above rules, it is clear that any string derived from `A` must ultimately start with beta. After the beta, the rest of the string must either be empty or consist of a sequence of one or more alpha's. Using a new nonterminal `A'` to represent the rest of the string we can represent the transformed grammar as follows:

```
A: beta A'
A':/* empty */
A': alpha A'
```

The above rules are not left recursive.

3.1.3 IAMC Math-Expression Grammar

The most important issue for using recursive descent parsing technology is to remove left recursive rules. In the mathematical expression rules, there are many left recursive rules such as

expression: expression '+' expression

After removing the left recursive rules according the above technology, we define the following rules which are currently supported by IAMC Client Grammar:

```
********* Simple IAMC Math-Expression Grammar  *********

IAMC-Math-Grammar
        : VAR ':'  math-expression
        : math-expression

math-expression
        : SIGN math-expression
        : VAR exprRest
        : NUM exprRest
        : '('  math-expression ')'  exprRest
        : OPS   '('  math-expression   ')'   exprRest
        : 'factor'   '('  poly     ')'    exprRest
        : 'expand'   '('  poly     ')'    exprRest
        : 'diff'       '('  math-expression  ',' VAR diffRest exprRest
        : 'integrate'  '('  math-expression  ',' VAR intRest  exprRest
        : Unknown-OP    '(' math-expression  argRest  ')' exprRest

OSP
        : 'sin' | 'cos'  | 'tan'  | 'cot'  | 'asin' | 'acos'
                | 'atan' | 'acot' | 'exp'  | 'log'
Unknown-OP
        : VAR

argRest
        : EMPTY
```

```
        : ','   math-expression  argRest

exprRest
        : '+'    math-expression
        : '-'    math-expression
        : '*'    math-expression
        : '/'    math-expression
        : '^'    math-expression
        : EMPTY

poly
        : SIGN poly
        : VAR polyRest
        : INT polyRest
        : '('   poly     ')'  polyRest

polyRest
        : '+'  poly
        : '-'  poly
        : '*'  poly
        : '^'  INT polyRest
        : EMPTY

diffRest
        : ')'
        : ','    NUM    ')'

intRest
        : ')'
        : ','    NUM    ','     NUM     ')'

SIGN
        : '+'
        : '-'

NUM
        : INT
        : REAL
        : 'pi'
        : 'e'
        : 'i'

VAR
        : letter varRest

varRest
        : letter varRest
        : digit  varRest
        : EMPTY

********** End of Grammar  **********
```

3.1.4 Tokens

Tokens are separated by token separators. Space, “\t”, and “\n” are always token separators, but “+”, “-”, “*”, “/”, “^”, “(”, “)”, “,” , and “:” are considered both tokens and token separators. So, in the expression *x-y+z* there are 5 tokens even there is no space in between at all. The `SIGN` token, `POSITIVE` or `NEGATIVE`, should be distinguished from `PLUS` and `MINUS` since they are the same representation. If it is a `SIGN` (`POSITIVE` or `NEGATIVE`), it must satisfy the following conditions:

1). There is no space between the `SIGN` and the following expression

2). The only allowed token before the `SIGN` is “(”, “,”, or “:”.

3). Otherwise it is NOT the `SIGN` token.

So in the following expression:

-x*y+2

`-` is `NEGATIVE`, `+` is `PLUS`. In

-x-(-y-x)

the first and the third `-` are `NEGATIVE` signs, the second and the forth are `MINUS` tokens.

3.1.5 Internal Representation

For the valid infix user input which was parsed successfully, IAMC has an internal representation in prefix format. All mathematical expressions will be kept in prefix notation and have the following format

(*operator operand1 operand2 ... operandN*)

The operands *operand1 operand2 ... operandN* can be subexpressions. One example is the expression

factor(x^2-y^2)

The internal representation for the above expression will be

((factor) ((-) ((^) x 2) ((^) y 2)))

The reason for putting operator into ()s is that it is easy to extract the operator; also we can add more information into the ()s besides the operator name. There are two reasons that we chose prefix as internal representation. Firstly, it is very close to MP data format. MP parses all data in the form of annotated parse trees. All data are exchanged as linearized annotated parse trees. A linearized version of the annotated tree is formed from a prefix parse of the tree. Node packets are distinguished from annotation packets by their position in the stream of packets. The second reason we chose prefix as our internal representation is that it is very easy to implement the 2-D display; the prefix representation makes it easy to separate the operator and operands.

3.1.6 Prefix Notation Generation

All operators have a pre-assigned operator precedence [16]. The operator `^` has higher precedence than operator `*` and operator `/`; operator `*` and `/` have higher precedence than operator `+` and operator `-`; all the other operators have the highest precedence and are treated as unary operators.

To generate prefix notation, two stacks `A` and `B` are used. Stack `A` holds operators; stack `B` holds operands. The following action rules are used in parsing and generating prefix notation:

1. An incoming operand is pushed onto Stack B.

2. If the incoming operator has higher precedence than the top operator's precedence on Stack A (or Stack A is empty), push the incoming operator onto stack A. Otherwise, pop the operator(s) in Stack A, pop the associated operand(s) in Stack B, generate the prefix notation, and push the prefix result back onto Stack B.

3. When the incoming token is "(", push it into Stack A, (the special "(" has lowest precedence), and also push it onto Stack B to mark the beginning of an expression.

4. When the incoming token is ")", all the operators before the ")" in Stack A should be popped and the associated operands in Stack B also should be popped to generate prefix notation. Push the prefix result into Stack B, pop up "(" from Stack A; then look at the top operator in Stack A; if it is a unary operator (except NEGATIVE or POSITIVE sign), it will also be popped and the prefix notation is constructed.

5. If the incoming token is ",", all the operators before the "(" in Stack A should be popped and the associated operands in Stack B also should be popped to generate prefix notation. Push the prefix result into Stack B.

Suppose the user inputs

```
x*2-sin(x+y)/y*z
```

the final prefix notation will be

```
((-) ((*) x 2) ((*) ((/) ((sin) ((+) x y)) y) z))
```

Figure 5 shows the steps of stack information during the prefix notation generation.

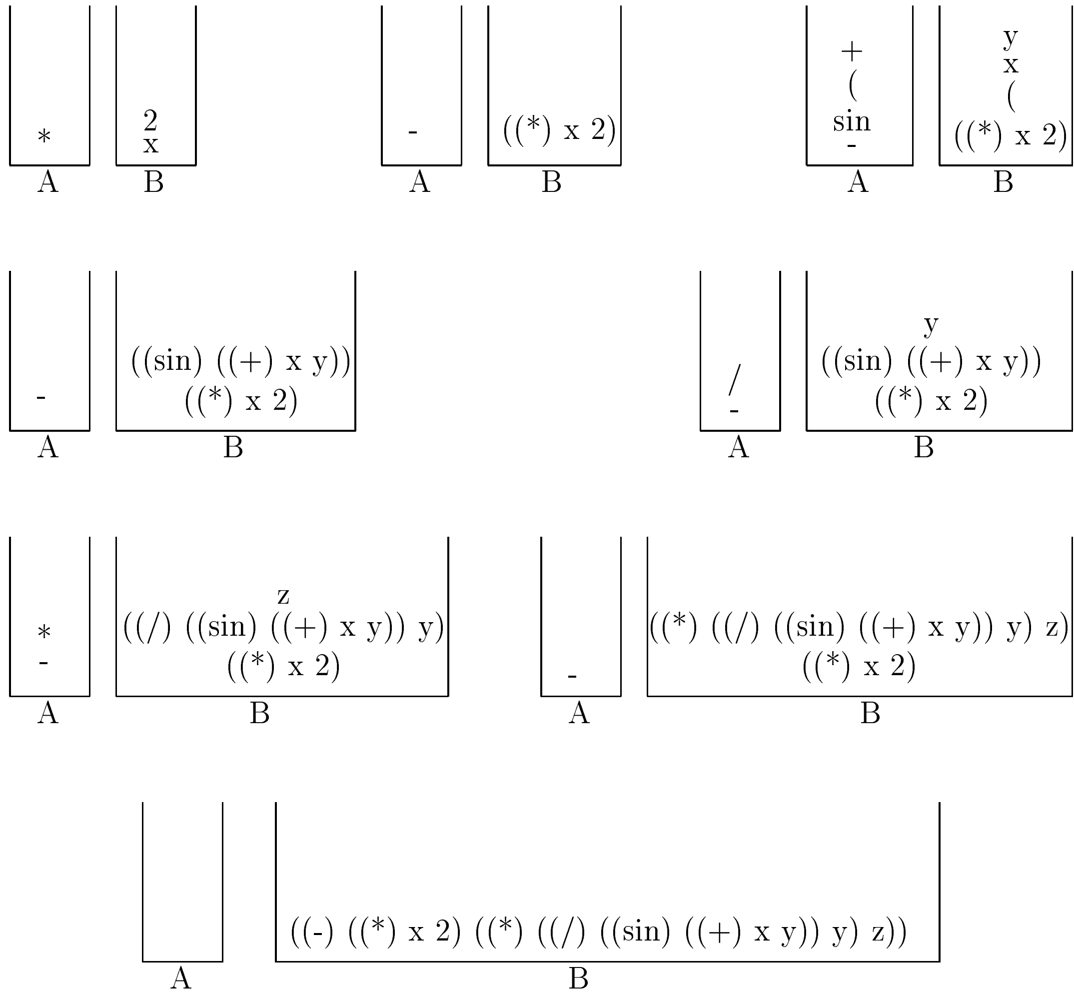

Figure 5: Example of Prefix Generating

3.2 ASCII–MP Conversions

MP is used for efficient exchange of data between scientifically oriented software tools. Mathematical data represented in ASCII are translated to MP before transmission. MP data received also need to be converted into other formats.

3.2.1 Prefix to MP Conversion

The IAMC Client stores input in prefix form. Before being sent to the remote IAMC Server, prefix data needs to be converted to MP. MP is supported by a set of C library functions. Thus, the ASCII to MP conversion is also written in C. The IAMC Client calls the Java `runtime` method to invoke the `prefix2mp` process.

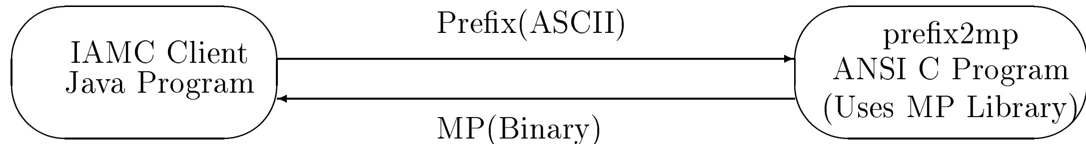

Figure 6: Java Exec Method

When the Java Virtual Machine runs, an instance of the class `Runtime` records the status of the running system and provides operations it can perform. The static method call

```
Runtime rt = Runtime.getRuntime();
```

returns the current `Runtime` object `rt`. It can initiate other programs on the platform from Java. Create a process, an instance of `Process`, with an `exec` method call:

```
Process p = rt.exec (command);
```

The *child process* [7] created runs independently on the host platform. The `Process` object allows the program to control and communicate with the child process. Figure 6 shows the relation between the Java program and the C program `prefix2mp`. IAMC Client passes the prefix string to the `prefix2mp` program. The `prefix2mp` function's duty is to construct an MP parsing tree according to the input.

The `prefix2mp` program uses MP defined operations such as `IMP_PutSint32` to perform its duty.

3.2.2 MP to ASCII Conversions

The result coming back from the IAMC server, or from an MP file that a user opens, is displayed by converting it to an ASCII format: prefix, infix, or LaTeX. This is simply done by using an existing prototype program `displaymp`, written in ANSI C, which takes an MP input and converts it to prefix, infix, or LaTeX as requested. Again the Java `exec` mechanism is used. By default, results are displayed with `displaymp -infix`. Commands are provided in the IAMC client to display any given result in the other two formats.

3.3 Interface to IAMC Server

In the IAMC Client part, the interface between IAMC Server and Client are implemented by three methods in the IAMCClient class:

1. **beginSession:** simply asking for a connection to a particular IAMC Server.
2. **remoteCompute:** send compute request in MP format to the server and get the answer back in MP format.
3. **endSession:** close the connection with the server.

Meanwhile, the class `WriteToFile` performs the duty of writing the answer coming from the server to a temporary MP file without the ending marker "ENDMP".

CHAPTER 4

IAMC Server

The experimental IAMC Server is a Java program invoked by the `inet` daemon. As such, it communicates with the IAMC Client through standard input and standard output. The evolving IAMC design calls for MCP, the *Mathematical Computation Protocol*, as the sever-client protocol. The MCP specification is on-going and promises to support multiple mathematical encoding schemes, among other features. However, the experiment here simply assumes each message between the server and the client to be MP-encoded.

The design of the IMAC Server consists of two major parts:

1. client interface.

2. external compute engine control.

These two parts will be described separately. Modifications to the compute engine for connecting to the IAMC Server are also described.

4.1 Interface to IAMC Client

The `IAMCServer` class controls the interface to the IAMC Client. When IAMC Server is invoked it starts the background compute engine, then waits for compute requests from the client (via standard input). The `WriteToFile` class will be re-used here. When the server reads data from client, `WriteToFile` will write such data into a temporary MP file and delete the ending marker "ENDMP". The method `getans`

of IAMCServer will collect the compute answer from compute engine, convert it to MP data, then send it back to client. When IAMC Client closes its socket, the Server will get the end of file indication.

4.2 The Compute Engine

The IAMC Server controls a MAXIMA [12] system running under UNIX. MAXIMA is a version of MACSYMA [23], a computer algebra system originally developed by the MATH Lab Group at MIT. The MAXIMA used is in Common Lisp (CL) with enhancements from the University of Texas at Austin and Kent State University. MAXIMA offers a wide range of capabilities, including differentiating and integrating expressions, factoring polynomials, plotting expressions, solving equations, manipulation of matrices, and computing Taylor series. MAXIMA is also a programming environment in which the user can define mathematical procedures tailored to his or her own needs. It can work with symbols, polynomial expressions, equations, and numbers. It allows the widespread use of many mathematical operations with various data types such as integer, rational, various sizes of floating points, and complex numbers. It also supports the use of irrational numbers such as “pi”. Many higher level data types and operations are also available including polynomials and matrices. MAXIMA is written in AKCL LISP. This allows users to develop their own high-level MAXIMA functions that call other high-level MAXIMA functions, MAXIMA data types, or LISP level functions. Any MAXIMA level function can be accessed at LISP level, and functions written at LISP level can be accessed at the MAXIMA level.

4.3 Interface to Compute Engine

Since MAXIMA is a local program, the IAMC Server just calls the Java `Runtime` method to execute the MAXIMA program in the background. The IAMC Server maintains two sets of input/output for different purposes: one is from/to the IAMC Client, the other is from/to MAXIMA program. The IAMC Server gets data (in MP format) from IAMC Client, converts it into MAXIMA input, sends it to MAXIMA, collects the results from MAXIMA, translates them into MP format, and sends them back to IAMC Client.

MP is the data format for IAMC Client and IAMC Server communication. One of the IAMC Server's responsibilities is to act as an adapter between compute engine and MP. Each different compute engine requires different treatment. Thus the server-to-engine interface is custom for different engines. When IAMC Server gets the MP data from IAMC Client, it should translate such MP requests to the compute engine input format, then send them to the compute engine. For MAXIMA, the input should be in infix ASCII format which is Fortran-like syntax followed by `;` as the command terminator. For easy experimentation, the IAMC Server invokes a child process `displaymp -infix` to convert the MP request from the Client to an infix format string, attaches a semicolon to it, then sends the resulting ASCII string to MAXIMA as input.

Handling MAXIMA output is a little more complicated. Normally, the MAXIMA top level is a read-eval-print loop much like that of lisp [12]:

1. The next command typed by the user is read.
2. The command is parsed and converted to internal form.
3. This internal form is evaluated by the function `meval`, the MAXIMA evaluator.

The `meval` execution produces a result which is in valid MAXIMA internal representation.

4. The result is simplified by the function `simplifya` which transforms the answer and returns a valid internal representation.

5. The final answer is displayed in 2-D form by `displa`.

6. The prompt for the next C-line is displayed and the cycles restarts from step 1.

So, when we issue the command *factor(x^3-1);* to MAXIMA, after the `displa` function the output will look like:

```
(C1) factor(x^3-1);
                                            2
(D1)                           (X - 1) (X  + X + 1)
```

The above output is useless for the IAMC Server, because the structure of the output is lost. The Server needs the un-rendered result from MAXIMA for converting to MP data. This means a slight modification to MAXIMA for the purpose of being controlled by the IAMC Server. We expect a certain amount of minor modification to any existing compute engine before it can be made into a part of IAMC.

The following simple Lisp program `newdispla.lsp` does the job:

```
;;;;;;;;;;;;;;;;;; File:  newdispla.lsp  ;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Purpose:   Makes maxima display prefix internal form
;;
;; Usage:     load this file into maxima
;;                loadfile("newdisplay.lsp");
;;
;; Author: Paul S. Wang
;; Date: 2/3/98
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package 'maxima)
(defun displa(exp)
     (print 'BEGIN-EXP)
```

```
    (print exp)
    (print 'END-EXP)
    (terpri)
)
```

The file `newdispla.lsp` should be loaded into MAXIMA first to redefine the `displa` function. The modification causes MAXIMA to return the internal prefix representation as final output for any computation. The above example now becomes

```
(C2) factor(x^3-1);

BEGIN-EXP
((MLABLE) $D2
 ((MTIMES SIMP FACTORED) ((MPLUS SIMP IRREDUCIBLE) -1 $X)
  ((MPLUS SIMP IRREDUCIBLE) 1 $X ((MEXPT SIMP RATSIMP) $X 2))))
END-EXP
```

Now the IAMC Server-to-MAXIMA interface can convert this data format into MP. A C program `maxima2mp` is written for this purpose. The `maxima2mp` program is similar to `prefix2mp`. The only difference is `maxima2mp` needs to convert the internal MAXIMA representation for operators and identifiers to the regular ASCII form. Basically it strips the "$" sign and converts operators and identifiers appropriately. For example, `MTIMES` becomes `*`, and `$X` is `X`. `$D2` is an identifier that labels the whole expression. Subsequent user commands can use `D2` to recall the expression. Thus, the above MAXIMA output should be converted to the following prefix ASCII expression representation:

```
((assign) D2
((*) ((+) -1 x) ((+) 1 x ((^) x 2)))
)
```

Such modified prefix data can then be translated to MP data using the same techniques as `prefix2mp`. Figure 7 shows the structure of the interface to the external engine.

Another duty of the interface is to convert the commands from MP form to the corresponding form for the compute engine. For example, in MAXIMA, the `INTEGRATE` operator has the name *integrate*, while in Maple, the name is *int*. An IAMC Server for Maple should convert *integrate* to *int* before sending it to Maple.

4.4 Server-Client Communication

IAMC Client and IAMC Server use stream socket for communications. A socket is a software mechanism representing a communication's entry point on the Internet. Since IAMC Client and IAMC Server are running on different host computers, independent processes must be able to initiate and/or accept communication requests in an asynchronous manner. A Client process uses its own socket to communicate with another socket belonging to a server process. Each host computer on the Internet creates its own sockets to communicate with other sockets on the Internet. A socket address consists of the numerical IP address of the host computer and an integer port number. Different types of sockets support different protocols. Communication takes place between sockets of the same type. Two important types are `Stream` socket and `Datagram` socket [8]. IAMC uses the stream socket. With a stream socket, a process can dial another stream socket's address and make a connection. A pair of connected stream sockets supports bidirectional, reliable, sequenced, and unduplicated flow of data without record boundaries. Stream sockets use the Transmission Control Protocol (TCP/IP).

The Java `Socket` and `ServerSocket` classes provide the client- and server-side stream socket mechanisms. In Java, a socket is the basic object in Internet communication, which uses the TCP protocol. TCP is a reliable stream network connection. The `Socket` class provides methods for stream I/O, which make reading from and

writing to a socket easy. `ServerSocket` is an object used for Internet server programs that listen to client requests. ServerSocket does not actually perform the service; instead, it creates a socket object on behalf of the client. The communication is performed through that object. Sockets are based on a client/server model. One program (the server) provides the service at a particular IP address and port. The server listens for service requests. Any program (client) that wants to be serviced needs to know the IP address and port to communicate with the server [7].

An advantage of the socket model over other forms of data communication is that the server doesn't care where the client requests come from. As long as the client is sending requests according to the TCP/IP protocol, the requests will reach the server, provided the server is up and the Internet isn't too busy. This also means that the client can be any type of computer. No longer are we restricted to UNIX, Macintosh, DOS, or Windows platforms. Any computer that supports TCP/IP can talk to any other computer that supports it through this socket model. This is a potentially revolutionary development in computing. Instead of maintaining armies of programmers to port a system from one platform to another, we write it once, in Java. Any computer with a Java virtual machine can run it.

The current IAMC server is on host `ox.mcs.kent.edu` and uses port 4450 for experiments. The IAMC Server first creates a server-side stream socket with

```
Serversocket listen = new ServerSocket(port);
```

on ox at the port number 4450. The socket is used to listen and wait for incoming connections:

```
Socket s_soc = listen.accept();
```

A new Socket object is returned when an incoming connection is made, that is, when an IAMC Client requests connection. After the connection is made, the Client and Server will communicate with each other. The Client will send the MP data to the Server and attach a string marker "ENDMP" to each MP request to indicate that one MP request is completed. When the Server receives the "ENDMP" marker, it knows that is the end of this MP data. Then the Server can send such MP data (excluding "ENDMP") to the `displaymp` program for conversion. When the server gets the answer from the MAXIMA and converts it to MP, it will send the data back to the Client. Also it will attach the same marker to the end to let the Client know where the MP data ends. Finally, when the IAMC Client decides to end such a computation session, it will close the I/O and the socket. The IAMC Server side will then automatically end this connection.

A better way to manage the server is to register it with `inetd` (inet daemon) (in the configuration file `inetd.conf`). This way the server does not have to set up its own socket but just uses standard I/O instead. This is because `inetd` will arrange a stream socket connection with the incoming client, redirect standard I/O to the stream socket, then fork the IAMC Server.

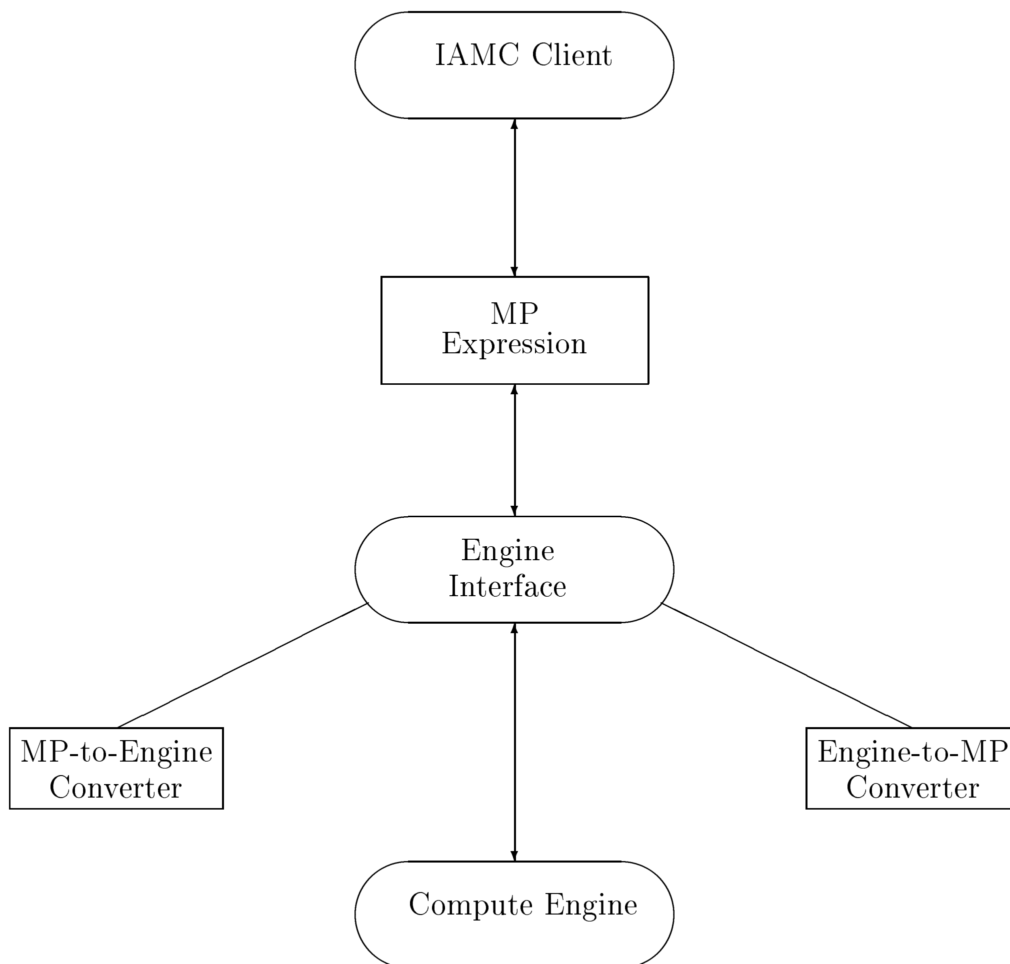

Figure 7: Interface to Compute Engine

CHAPTER 5

IAMC Usage Examples

The IAMC system is an ASCII menu-driven application. The following examples show the actual scripts of using IAMC for several computations. Each example demonstrates a different aspect of the IAMC system.

5.1 Example 1

The purpose of this example is to demonstrate the menu of IAMC Client and how to connect with IAMC Server. Also it shows how the operator precedence issue can be correctly recognized and the correct answer of numeric computation can be obtained from the IAMC Server.

```
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
*   1). Type your infix input   2). New Session                *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
Select (1-11):2
Server:ox.mcs.kent.edu MAXIMA connected.
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
*   1). Type your infix input   2). New Session                *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
```

```
Select (1-11):1
Please input your expression:
-19*98^2+1998/2-(-1998-1998)
Valid input.
*****************************************************************
*                    IAMC-Client  MENU                          *
*---------------------------------------------------------------*
*   1). Type your infix input   2). New Session                 *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                     *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                        *
*  11). Exit                                                    *
*****************************************************************
Select (1-11):4
d2: -177481
*****************************************************************
*                    IAMC-Client  MENU                          *
*---------------------------------------------------------------*
*   1). Type your infix input   2). New Session                 *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                     *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                        *
*  11). Exit                                                    *
*****************************************************************
Select (1-11):5
Server:ox.mcs.kent.edu MAXIMA closed.
```

5.2 Example 2

This example shows that IAMC can also handle real numbers, and the special constants. Also it tests the supported operators `sin`, `log`, and `sqrt`.

```
*****************************************************************
*                    IAMC-Client  MENU                          *
*---------------------------------------------------------------*
*   1). Type your infix input   2). New Session                 *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                     *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                        *
*  11). Exit                                                    *
*****************************************************************
Select (1-11):1
Please input your expression:
sin(pi)+log(e^2)+sqrt(19.98)
Valid input.
```

```
************************************************************
*                    IAMC-Client  MENU                     *
*----------------------------------------------------------*
*   1). Type your infix input   2). New Session            *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                   *
*  11). Exit                                               *
************************************************************
Select (1-11):4
d2:          6.469899178
```

5.3 Example 3

This example has two purposes: one is to show that IAMC can handle unsupported operators such as `sum` and get the correct answer back; the other is to show how IAMC deals with user input error. That is, IAMC will show where it is stopped (the next token is wrong in the input).

```
************************************************************
*                    IAMC-Client  MENU                     *
*----------------------------------------------------------*
*   1). Type your infix input   2). New Session            *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                   *
*  11). Exit                                               *
************************************************************
Select (1-11):1
Please input your expression:
sum(x^2+x, x, 1, 10)
Valid input.
************************************************************
*                    IAMC-Client  MENU                     *
*----------------------------------------------------------*
*   1). Type your infix input   2). New Session            *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                   *
*  11). Exit                                               *
************************************************************
Select (1-11):4
```

```
d2: 440
*************************************************************
*                   IAMC-Client  MENU                       *
*-----------------------------------------------------------*
*   1). Type your infix input   2). New Session             *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                 *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                    *
*  11). Exit                                                *
*************************************************************
Select (1-11):1
Please input your expression:
sin(x+cos(x+y1/z1*(kk^2)))-integrate(x^9,x,1,,4)
Invalid input, parse stopped at:
sin(x+cos(x+y1/z1*(kk^2)))-integrate(x^9,x,1,
```

5.4 Example 4

This example is meant to show in one session how the user can do computation, make variable assignment, and use label representation. Also it demonstrates that IAMC can do symbolic computation, keep the context of one session, and recognize the naming of an expression and label representation. The symbolic computation operators `factor`, `integrate`, `expand`, and `diff` will also be tested.

```
*************************************************************
*                   IAMC-Client  MENU                       *
*-----------------------------------------------------------*
*   1). Type your infix input   2). New Session             *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                 *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                    *
*  11). Exit                                                *
*************************************************************
Select (1-11):1
Please input your expression:
y:9*x^2-1
Valid input.
*************************************************************
*                   IAMC-Client  MENU                       *
*-----------------------------------------------------------*
*   1). Type your infix input   2). New Session             *
*   3). Send MP file to Server  4). Send current exp to Server *
```

```
*   5). End Session              6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
Select (1-11):4
d2: -1 + 9*x^2
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
*   1). Type your infix input   2). New Session                *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session              6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
Select (1-11):1
Please input your expression:
factor(y)
Valid input.
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
*   1). Type your infix input   2). New Session                *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session              6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
Select (1-11):4
d3: (-1 + 3*x)*(1 + 3*x)
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
*   1). Type your infix input   2). New Session                *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session              6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
Select (1-11):1
Please input your expression:
integrate(d3,x,1,4)
Valid input.
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
*   1). Type your infix input   2). New Session                *
```

```
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
Select (1-11):4
d4: 186
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
*   1). Type your infix input   2). New Session                *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
Select (1-11):1
Please input your expression:
expand(d3)
Valid input.
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
*   1). Type your infix input   2). New Session                *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
Select (1-11):4
d5: -1 + 9*x^2
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
*   1). Type your infix input   2). New Session                *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                    *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                       *
*  11). Exit                                                   *
****************************************************************
Select (1-11):1
Please input your expression:
diff(d5,x)
Valid input.
****************************************************************
*                    IAMC-Client  MENU                         *
*--------------------------------------------------------------*
```

```
*   1). Type your infix input   2). New Session              *
*   3). Send MP file to Server  4). Send current exp to Server *
*   5). End Session             6). History                  *
*   7). Save current exp to MP  8). Save current result to MP  *
*   9). Open MP file           10). Help                     *
*  11). Exit                                                 *
****************************************************************
Select (1-11):4
d6: 18*x
```

CHAPTER 6

Conclusion

Work reported here is part of the overall IAMC effort being conducted by Wang's research group at ICM/Kent. This work builds an experimental client-server package for a bare-bones version of an IAMC system, and the system does serve to prove the concept and to expose many details for further investigation.

Programming work reported here includes the Java-coded client and server parts, i.e., all the classes and functions in the IAMC Client and Server, as well as the ANSI C coded MP conversion programs `prefix2mp` and `maxima2mp`. Alos Gray's MP library and Wang's `displaymp` are used on the client and server parts. It is hoped that the programming here can provide a starting point for modification, improvements, and further development of IAMC.

Also, the current system does not take full advantage of MP. Not all MP supported data formats can be converted yet. This is just a matter of extending the program to handle all cases.

Future work on IAMC includes a GUI on the Client side, full specification of MCP, Java coded MCP classes to be used on the Client and the Server sides, making IAMC compatible and accessible via the Web and via email, supporting multiple mathematical encoding formats, handling graphing/plotting, providing help and documentation information for the user, aborting computations, and applying MCP in problem solving environments.

It is the hope of the author that this work has provided some ground work for future research in the area of Internet Accessible Mathematical Computation.

BIBLIOGRAPHY

[1] Doleh, Y. K. and Wang, P. S. "SUI: A System Independent User Interface for and Integrated Scientific Computing Environment," Proceedings of ISSAC'90, Addison-Wesley 1990.

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

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

[4] Kajler, N. and Soiffer, N. "A Survey of User Interfaces for Computer Algebra Systems," Journal of Symbolic Computation 11, 1995.

[5] Young, D. A., Wang, P. S. "GI/S: A Graphical User Interface for Symbolic Computation Systems," Journal of Symbolic Computation 4, 1987.

[6] Gray, S., Kajler, N,, and Wang, P. S. "Design and Implementation of MP, a Protocol for Efficient Exchange of Mathematical Expressions," Journal of Symbolic Computation 11, 1996.

[7] Wang, P. S. *Java with OOP and Web Applications*, Brooks/Cole Publishing Co., Pacific Grove, CA, ISBN 0-534-95206-2, 1998

[8] Wang, P. S. *An Introduction to UNIX with X and the Internet*, PWS Publishing Co., Boston, MA, ISBN 053494768-9, July 1996.

[9] Sams.net Publishing *Internet Unleashed*, 2nd edition, ISBN 0-672-30714-6, 1995.

[10] Newman, A. *Using Java*, Que Publishing, ISBN 0-7897-0604-0, 1996.

[11] Symbolics, Inc. *Macsyma User's Guide*, January 1988.

[12] Wang, P. S. "Helpful Hints to MAXIMA Programmers," (class handout).

[13] Doleh, Y. K. "The Design and Implementation of a System Independent User Interface for an Integrated Scientific Computing Environment," Ph.D. Dissertation, Kent State University, May 1995.

[14] Recursive Descent Parsing "`http://opal.cs.binghamton.edu/~zdu/`"

[15] Ion, P. and Miner R. "Mathematical Markup Language," W3C Working Draft, January 6, 1998.

[16] Aho, A. V., Sethi, R., and Ullman, J. D. *Compilers Principles, Techniques, and Tools*, Addison-Wesley Publishing, ISBN 0-201-10088-6, 1985.

[17] Smith, C. J. and Soiffer, N. M. "MathScribe: A User Interface for Computer Algebra Systems," Proceedings 1986 Symposium on Symbolic and Algebraic Computation, pp.7-12, ACM Press 1986.

[18] Arnon, D., Waldspurger, C., McIsaac, K. "CaminoReal User Manual," Version 1.0, Technical Report CSL-87-5, Xerox PARC, 1987.

[19] Paracomp, Inc. *Milo User's Guide*, 123 Townsend St., Suite 310, San Francisco, CA 94107, 1988.

[20] Purtilo, J. M. "Polylith: An Environment to Support Management of Tool Interfaces," ACM Sigplan Notice, 20(7): 12-18, July 1985.

[21] Char, B. W., Geddes, K. O., Gonnet, G. H., Leong, B. L., Monagan, M. B., Watt, S. M. *Maple V Language Reference Manual*, Springer-Verlag. ISBN 0-387-97622-1, 1991.

[22] Jenks, R. D. and Sutor, R. *AXIOM, the Scientific Computation System*, Springer-Verlag, 1992.

[23] Genesereth, M. R. "An Automated Consultant for MACSYMA," Proceedings 1977 MACSYMA Users' Conference, pp. 309-314, 1977.

[24] Leong, B. "Iris: Design of an User Interface Program for Symbolic Algebra," Proceedings 1986 Symposium on Symbolic and Algebraic Computation, pp.1-6, ACM Press, July 1986.

[25] MathSoft, Inc. "MathStation," Version 1.0 (a computer program). 201 Broadway, Cambridge, MA, 02139, 1989.

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

[27] Pintur, D. A. "MathEdge: The Application Development Toolkit for Maple," MapleTech 1(2), pp. 31-38, 1994.

[28] von Sydow, B. "The design of the Euromath system," Euromath Bulletin 1(1), pp. 39-48, 1992.

[29] Diaz, A., Kaltofen, E., Schmitz, K., Valente, T., Hitz, M., Lobo, A., and Smyth, P. "DSC: A System for Distributed Symbolic Computation," Proceedings of ISSAC'91, Bonn, Germany, pp. 323-332, ACM Press, July 1991.

[30] Dalmas, S., Gaëtano, M., and Sausse, A. "ASAP: a protocol for symbolic computation systems," INRIA Technical Report 162, March 1994.

[31] Abbott, J., and Traverso, C. "Specification of the POSSO External Data Representation," Technical report, Sept. 1995.

[32] Wolfram, S. *Mathematica: A System for Doing Mathematics by Computer*, Addison-Wesley, 1988.

[33] Wolfram Research, Inc. "MathLink Reference Guide," Mathematica Technical Report, 1993.

[34] Bonadio, A. and Warren, E. *Theorist Reference Manual*, Prescience Corporation, 814 Castro St., San Francisco, CA 94114, 1989.

[35] XML "`http://www.arbortext.com/xml`".

[36] SGML "`http://www.sil.org/sgml/sgml.html`".

[37] HTML "`http://www.w3.org/TR/`".

[38] Calculators "`http://www-sci.lib.uci.edu/HSG/RefCalculators.html`".

[39] Fitch, J. P. *CAMAL User's Manual*, University of Cambridge Computer Laboratory, 2nd edition, 1983.

[40] Greuel, G. M., Pfister, G., and Schönemann, H. "Singular Reference Manual," Reports On Computer Algebra, number 12, Centre for Computer Algebra, University of Kaiserslautern, May 1997.