Web-based Mathematics Education with MathChat

David Chiu
Institute for Computational Mathematics
Department of Computer Science
Kent State University, Kent, OH 44242-0001
dchiu@cs.kent.edu

Abstract

MathChat is a protocol to support web-based chat sessions to simulate a classroom-type environment enabled for mathematics education. With MeML support, MathChat allows mathematics computation and display, which empowers mathematics educators to hold lectures and discussions outside the physical classroom. An online educational framework offers such advantages as more flexibility in scheduling and less apprehension to "shy" students.

1. Introduction

The Web-based Mathematics Education Framework (WME) [17] [18] discusses the delivery of mathematics lessons as web pages through the use of MeML [16] [19] and interaction between these documents and compute engines with the MeML processor [17]. We now discuss the basis for chat support in the WME framework since "it is important that learners be able to interact with each other, with resources of instruction, and with their teacher" [15]. A case study by The American Youth Policy Forum has also shown that students are less apprehensive on class participation in an online classroom environment [10].

Many enterprise Learning Management Systems (LMS) offer chat support as a part of their package: WebCT [9], TopClass [8], and Blackboard [1] just to name a few. While the WME Framework is not exactly a software package (but rather a distributed learning system for mathematics), we feel that support for live interaction between educators and students is equally essential. In spite of this, we sought to select from existing chat protocols for our framework. But by analyzing their specifications and uses, we came to a fickle conclusion: the existing protocols offer too little yet too many functionalities for our use.

Internet Relay Chat (IRC) [7] is perhaps the most ubiquitous of all chat protocols. Even prior to its formal conception in 1993, it has been, and still stands as the foundation for a widely accessible public-use chat. But because it was designed for general socialization, it has no notion for supporting education (let alone mathematics education). Its chat rooms (known as channels) can be manipulated to simulate a classroom environment, but we feel the extra work to learn the many types of channel modes and functionalities is unnecessary and irrelevant when it comes to teaching mathematics. Furthermore, because IRC is based on plain text, mathematical display is limited to infix. In conclusion, the IRC protocol is basically too general and superfluous when we are in search of a very specific classroom atmosphere.

With the Java-based TechTalk [12], developed at Drexel University, chat is done through a separate window from a shared mathematics computation screen supporting Maple [3] and MATLAB [5] sessions. TechTalk incorporates logging capabilities for referencing after a session is completed. While this is similar to MathChat's goals, it still did not meet our requirements. MathChat seeks to support both chat and mathematical expressions in the same window by using MeML. MeML includes the use of XHTML [2] and MathML [4] for structural content and mathematics display. MeML is also independent from specific compute engines such as Maple and MATLAB. Ultimately, we found TechTalk to be too closely tied with predetermined compute engines while the WME Framework already has the means to communicate with arbitrary compute engines with the MeML processor, OMEI [13] [14], and XMEC [20].

At the 2003 IAMC Workshop, WMEchat [11] was presented as an initial implementation for the goals of MathChat. WMEchat, however, was ultimately a CGI application. Clients were typical web browsers and all message processing was done on the web server via server-side programming with PHP [6]. No messages were actually sent on top of HTTP since they were sent as HTTP POST/GET values. It was an ad hoc solution for teacher-student interaction, but drawbacks such as overhead and inefficiency called for specifications for a new protocol.

These systems, however, provide valuable resources and references. The MathChat protocol adopts conventions, terminologies, and functionalities from the above systems, in particular to IRC. We borrow from the IRC protocol (RFC 1459 and RFC 2812) extensively because of its availability, wide-use popularity, and proven stability.

2. MathChat's Goals

MathChat aims to incorporate chat and mathematical lesson elements into one interface, with MeML its primary support for mathematics display.

- Support for MeML elements.
- Multiple support for mathematical input (infix, MathML, MeML).
- Physical classroom environment simulation and opportunities to provide a better one.
- Ease of displaying mathematical expressions—With MathML support in Internet browsers, we easily overcome the difficulty of mathematical display.
- Simple integration—MathChat can be deployed along-side MeML pages by a server host.
- Easy database access—Many CGI and server-side languages natively support database access. A database can provides an efficient data storage for server control.
- Support of Mathematics—By using MeML, we can better support chat on mathematics.
- Extensibility—Chat related WME services can augment and extend the basic functonalities and features of MathChat.

3. Entities in MathChat

3.1. Client Identification

The MathChat Protocol follows the client-server model. The server must know where to direct messages. Thus, each client must be identified by a unique ID. In MathChat, the unique ID is the client's nickname and address in the following format: nickname!address. The address portion, of course, is the IP address or hostname (if available) of the user's origin. A typical unique ID resembles:

- *david!cs.kent.edu*
- *david!131.123.35.92*

This information must be readily available after connection establishment with the server for all subsequent message passing. But even before message passing, we need to assign the user with an nickname to complete the unique ID "pair".

3.2. Nicknames

The format of the unique ID will allow access for different users from different hosts as well as different users from the same host. But before the user can do anything productive, a nickname must be passed to the server. Initially, no one is actually registered with the server. Unregistered users default to nicknames in this form: GuestXXXXX, where XXXXX is some unique 5-digit integer. After initial login, the user can then change their nicknames and register another one that is more of liking. Nicknames are not completely arbitrary, as they do have some constraints:

- Are to be no longer than 16 characters.
- Are case-insensitive.
- While nicknames can contain alphanumerics, they must start with an alphabetical character.

That is, nicknames must match this regular expression: */[a-z]([a-z]|[0-9]){0,15}/i*

Once a user has registered a nickname (more on registration and other commands later), the client can send the the nickname/password pair to the server to avoid the arbitrary GuestXXXXX nickname. Because the GuestXXXXX nicknames are reserved by the server, they cannot be registered to a user.

3.3. Classrooms

Classrooms are where teachers can hold discussions and lectures. It is the equivalent to a "chat room" or "channel" in other chat implementations. The difference is that MathChat classrooms simulate real classroom environments and allows mathematics expression display through the use of MeML.

Classroom names, like nicknames, have some constraints:

- Are to be no longer than 16 characters.
- Are case-insensitive.
- Must begin with a # character.
- The 15 characters following the # prefix must be alphanumeric.

That is, classroom names must match this regular expression: */#([a-z]|[0-9]){0,15}/i*

MathChat classrooms has two modes: *lecture* and *discussion*. In lecture mode, students are generally not permitted to speak unless "called" upon. Students would then have the option to "raise" their hand for permission to ask a question or comment on something. In discussion mode, unlike lecture mode, the classroom would not be under moderation.

The user that first establishes the classroom is known as the teacher. The teacher of the classroom has many privileges to maintain order in his/her classroom. These include:

- Opening up his/her classroom for chat.
- Closing his/her classroom to end lecture/discussion.
- Toggling between lecture and discussion modes.
- Kicking a user out of the classroom.
- Banning a user from further joining the classroom.
- Picking on a user to speak during lecture mode.
- Granting other users *teacher* status in the classroom.

These privileges are actually a subset of commands available to all MathChat users. Messages and commands are discussed next.

4. Client-to-Server Messages

As mentioned above, because HTTP is not a connection-oriented protocol, a means to track the true origin of the message is absolutely necessary. Thus, all client-to-server messages must be prefixed with the unique ID. Also, because of the use of HTTP, we cannot rely on a steady stream of connection so message may become lost. Consequently, MathChat, unlike IRC and most chat protocols, is synchronous in communication. As clients send messages to the servers, they should always anticipate an acknowledgment or reply.

A client-to-server message consists of the following components: unique ID, command, and command parameters. A typical client-to-server message may resemble: *nickname!cs.kent.edu JOIN #Math101*

In this example, JOIN is the command, and #Math101 is its parameter. There are a number of commands available to the client, but if a client sends an unknown command to the server, it results in an error reply. The following are the commands available to the client. Please note that this is a very tentative and incomplete list. A full set of commands is still under investigation and are to be updated capriciously.

4.1. Connection Setup Commands

- USER [unique ID]—This is sent to the server immediately after connection to the server has been established and the welcome message is received by the client. Because the unique ID is dependent on a nickname, the client should have a means for the user to specify a nickname before the connection to the server is made. The server then checks to see if the nickname is already in use, and if so, changes the nickname to GuestXXXXX.
- MOTD—Requests for the server's Message Of The Day

4.2. Nickname and Classroom Service Commands

- NICK [new nickname]: Sets current nickname to [new nickname].
- NICKREG [password] [email]—Registers the nickname currently being used by the client. As stated previously, GuestXXXXX nicknames cannot be registered.
- CLASSREG [classroom] [password] [email]—Registered the classroom under nickname being used.
- CLASSCLOSE [classroom]—Closes a classroom session. This command is available only to users with *teacher* status in the classroom.
- IDENTIFY [nickname|classroom] [password]—While nickname identification is normally established in connection setup, if a user changes nicknames after the initial setup, he will have to identify before the server changes it to the arbitrary GuestXXXXX nickname. For the same reason, a user may need to identify to the classroom that he/she owns if the classroom was registered under a different nickname.
- PASSWD [nickname|classroom] [old password] [new password]—Changes the password to [new password] for the specified nickname or classroom.
- SENDPASS [nickname|classroom]—Prompts the server to send the password for the specified nickname or classroom to the email address specified during registration.
- SETEMAIL [nickname|classroom] [password] [new email address]—Changes email addresses for specified nickname or classroom.

4.3. Classroom Management Commands

- MODE [classroom] [moderate|discussion]—Moderate mode allows only the users with *teacher* or *voiced* status to speak. Discussion mode allows all users to speak. This command is only available to users with *teacher* status.
- KICK [classroom] [nickname]—Kicks a user out of the specified classroom. This command only is available to users with *teacher* status.
- BAN [classroom] [nickname|host]—Kicks a user out of the specified classroom and disallows the user or users from the specified host from joining the classroom until the ban is lifted. This command only is available to users with *teacher* status.
- SETTEACHER [classroom] [nickname]—Grants a user with *teacher* status in the specified classroom. This command only is available to users with *teacher* status.
- DELTEACHER [classroom] [nickname]—Removes *teacher* status from the specified user. This command only is available to users with *teacher* status.
- SETVOICE [classroom] [nickname]—Grants a user with *voice* status in the specified classroom. Voiced users can speak in a moderated classroom with no other privileges. This command only is available to users with *teacher* status.
- DELVOICE [classroom] [nickname]—Removes *voice* status from the specified user. This command only is available to users with *teacher* status.
- VREQ [classroom]—Requests a voice in the specified classroom. This is similar to "raising" one's hand in real life situations.
- BANLIST [classroom]—Queries for the list of banned members from the specified classroom.
- LOGLIST [classroom]—Queries for the list of logged classroom discussions.
- DELLOG [filename|all]—Deletes the specified classroom log file or all classroom log files.

4.4. Chat and Control Commands

- MESSAGE [nickname|classroom] [text]—Sends [text] to the specified destination. The [text] field can hold plain text and MeML elements. Remember that MeML allows XHTML, MathML, and MeML elements.
- WHOIS [nickname|classroom]—Queries the server for information on the specified nickname or classroom.
- JOIN [classroom]—Joins the specified classroom for discussion.
- PART [classroom] [text]—Parts the specified classroom with an optional message.
- LIST—Retrieves a list of available classrooms. Only occupied classrooms will be returned.
- NAMES [classroom]—Retrieves the list of names in the specified classroom.
- QUIT [text]—Ends chat session and disconnect from the server with an optional message.

The client implementation should provide some form of abstraction to hide information that are not particularly essential for the user to know. Conventionally, the above messages are sent with a '`/`' character followed by the command name and its parameters. For example,

- */mode #MathChat101 moderate*
- */message jack this is a private message to jack*

5. Server-to-Client Messages

A server-to-client message consists of the unique ID of the destination, a numerical reponse or error number, an optional user/classroom of relevance, and the corresponding message. A typical server-to-client message may look something like:

- *dchiu!cs.kent.edu 001 WELCOME Welcome to MathChat on cs.kent.edu*
- *dchiu!cs.kent.edu 401 ERR dchiu The USER command is only valid during connection setup*

Likewise to client-to-server commands, due to the current development of the abundant replies, it is not to possible display all possible server-to-client messages. Because of this, the numberings and other parameters are tentative or even arbitrated for understanding.

5.1. Connection Setup Replies

- 001 WELCOME Welcome to MathChat on [machine name]
- 002 CONNECT +ok
- 003 MOTD Message of the day:

- 004 MOTD [motd text]
- 005 MOTD End of MOTD
- ...

5.2. Nickname and Classroom Service Replies

- 101 NICK [new nickname] [old nickname] is now known as [new nickname]
- 102 NICKREG [nickname] has been registered under [unique ID]
- 130 CLASSREG [classroom] has been registered under [nickname]
- ...

5.3. Classroom Management Replies

- 201 BANLIST [classroom] [unique ID 1, unique ID 2, ...]
- 202 BANLIST [classroom] End of BANLIST
- 203 LOGLIST [classroom] [date] [url]
- 204 LOGLIST [classroom] End of LOGLIST
- ...

5.4. Chat and Control Replies

- 301 WHOIS [nickname] is [unique ID]
- 302 WHOIS [nickname] is on [classroom1, classroom2, ...]
- 303 WHOIS [classroom] is owned by [nickname]
- 304 WHOIS [classroom] is currently [open/closed] for discussion
- 305 WHOIS [nickname] no such nickname or classroom
- 306 WHOIS End of WHOIS
- 311 LOAD [URL]
- 313 JOIN [classroom] +create [filename]
- 314 JOIN [classroom] +load [URL]
- 315 JOIN [classroom] [nickname] +ok
- 316 QUIT [nickname] [text]
- 317 QUIT +ok
- 350 MESSAGE [source nickname] [target nickname/classroom] [text]
- ...

5.5. Error Messages

- 401 ERR [nickname] is already in use
- 402 ERR [nickname|classroom] is already registered
- 403 ERR [nickname] You need to be using a registered nickname
- 404 ERR [classroom] You must have TEACHER status to perform this command
- 405 ERR [classroom] Moderated. Cannot send to classroom.
- 406 ERR [nickname] User does not exist.
- ...

For instance, the WHOIS command might generate 301 through 306. The NICK command might generate the error, 401, if the desired nickname is already in use or 101 upon success.

6. Message Conveyance

All messages must be relayed by the server. From a client, to be able to send a messages to a classroom or another client, the message is sent through the server and the server transmits the message to the recipient(s) by the unique ID. Below is a typical scenario of a client-server interaction. Because HTTP is a session-based protocol, the connection breaks after each of the following steps. This is ultimately why the unique ID is essential for message relaying.

6.1. Handshaking

Before effective communication with the server can be achieve, a client must complete a small series of initialization. For our example, we will assume that our client, *nickname1* is connecting to a MathChat server at *http://cs.kent.edu/MathChat/*:

- The initial connection is made when *nickname1* connects to *http://cs.kent.edu/MathChat/*.
- The server responds with a welcome message.
- On receipt of this message, the client creates a local “server status” file (i.e. MathChat.meml).
- The MathChat client must now register itself with the server. This is done by sending the USER command such as: *USER nickname1!cs.kent.edu*.

- The server now determines if *nickname1* already exists. If so, the server assigns our client to a GuestXXXXX nickname, where, as mentioned before, XXXXX is some random sequence of numbers.

- Once the USER command is accepted, the server replies with *002 CONNECT +ok*. It is extremely important to note that at this stage, *all* messages are prefixed with the unique ID, *nickname1!cs.kent.edu*. Thus, the server actually replies with *nickname1!cs.kent.edu 002 CONNECT +ok*. In preventing repetitiveness, the following scenarios disgard the unique ID prefix in messages in favor for the actual semantically meaningful fragment.

- The initialization phase is now complete and *nickname1* becomes an existent entity on the MathChat server.

6.2. Handling Server-Personal Messages

Throughout a chat session, each user has its own perspectives on the chat session. For example, *nickname1* types */whois nickname2*. The server replies with *nickname2's* information. However, this information is only relevant to *nickname1*.

Information of this type should be stored in the client-side "server status" file that was created during the handshaking phase.

6.3. Initial Join of a Classroom

After the initial steps, the client may want to join or create a classroom. For the following scenario, we will assume that the classroom, #Math101 is non-existent on the server, and that *nickname1* is the first to join it.

- Our client sends the following message to request joining of a new classroom, called #Math101: *JOIN #Math101*

- Upon receipt, the server creates a dialogue file in this format: classroom_unixtime.meml (i.e. Math101_1066368408.meml). Notice that the # symbol has been stripped from the filename.

- Unixtime is simply used to distinguish between different sessions of the same classroom, while concurrently serving as an easy way of determining the time/date of the classroom session's conception.

- The server updates the dialogue file with the JOIN message.

- The server then replies with the following message: *314 JOIN #Math101 +load http://cs.kent.edu/MathChat/Math101_1066368408.meml*.

- Upon reception of the successful 314 reply, the client downloads the file Math101_1066368408.meml and displays it on its web browser. At this point, the JOIN operation is complete.

6.4. Subsequent Joins of a Classroom

Of course, more clients must join our classroom in order to uphold a practical discussion. The following shows a typical scenario for a client, *nickname2*, who subsequently joins #Math101.

- Upon receiving a JOIN request, the server first checks nickname2 for join permission (i.e., if the client's nickname or address is banned?). If banned, the server replies with an error.

- Otherwise, the server updates its dialogue file with the new JOIN message and transmits a message to all prior participants with: *315 JOIN #Math101 nickname2 +ok*

- Upon receipt of reply 315, the clients update their dialogue file with the new join message.

- Note that this time, the server does not have to process file creation. It simply sends this reply to *nickname2*: *314 JOIN #Math101 +load http://cs.kent.edu/MathChat/Math101_1066368408.meml*.

- Upon receipt of reply 314, *nickname2* downloads the dialogue file pointed by the URL in the message in order to receive the discussion in its entirety.

- The use for this feature is apparent. If important concepts were discussed before *nickname2's* arrival, the user will not lose out any valuable information as the entire conversation dialogue is transferred.

- It is common to use the NAMES command immediately after a successful join in order to retrieve the list of participants.

Figure 1 depicts the flow chart of decisions and processes within the server on a JOIN message.

6.5. Classroom Messaging

Once clients have joined our virtual classroom, #Math101, they may communicate with each other by utilizing the MESSAGE command.

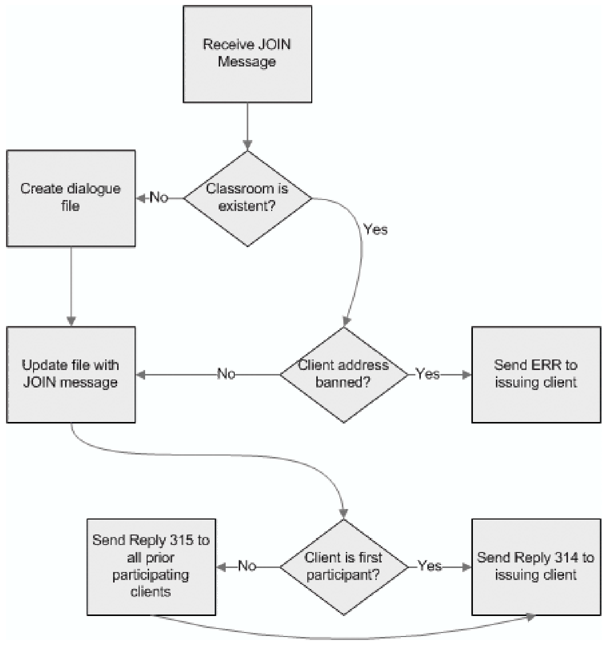

Receive JOIN Message
Classroom is existent?
No
Yes
Create dialogue file
Update file with JOIN message
Client address banned?
No
Yes
Send ERR to issuing client
Client is first participant?
No
Yes
Send Reply 315 to all prior participating clients
Send Reply 314 to issuing client

Figure 1. Internal Server Operations on the JOIN Message

- *Nickname1* sends *MESSAGE #Math101 hello all!* to the server in order to send the statement "hello all!" to the classroom.
- As server receives MESSAGE commands, it must first verify that the client is indeed "in" the classroom. If so, the server then determines if the client has privileges to speak in the specified classroom (e.g. if the classroom is moderated or if the client has the teacher or voice status). If the client does not have speaking permissions, the server transmits an error message.
- Otherwise, the message is placed in the tail of some message queue stored on the server.
- The message queue is then processed on a FIFO basis. With each message at the head of the queue, the server updates its copy of the classroom file with the message statement, then relays the statement to all clients associated with the classroom: *350 MESSAGE nickname1 #Math101 hello all!*
- Upon receipt, the clients update their local dialogue file with the message and the file is reloaded in their web browser to display the new message.

6.6. Parting Classrooms

In order to disconnect from the classroom, the client can either use the PART or QUIT command. Let us assume that our client, *nickname2*, wanted to leave #Math101 but to stay connected to the server.

- *Nickname2* sends *PART #Math101 bye!* to the server.
- When the server receives this message, it verifies that the client is indeed in #Math101. If not, an error message is sent to the client. Otherwise, the server immediately sends: *360 nickname2 PART #Math101 +ok* to *nickname2*.
- The server then updates its dialogue file with the PART message and transmits a message to all participants with: *361 nickname2 PART #Math101 bye!*
- Upon receipt of reply 361, the participating clients update their dialogue file with the PART message.
- At this time, the PART command is complete and *nickname2* no longer receives messages from the server with regards to #Math101. Furthermore, subsequent MESSAGE commands with #Math101 as its destination will be denied by the server.

6.7. Private Messaging

Clients can also message other clients directly (though, "directly" is deceptive, as this protocol orders that all messages be conveyed by the server). Abiding convention, we will call these client-to-client messages *private messages*.

- Our client, *nickname1*, sends the following message to request the creation of a private message session to *nickname2*: *MESSAGE nickname2 hello nickname2!*
- Unlike classroom messaging, the server need not create a file for the dialogue because the dialogue history should be initiative to both clients.
- Upon receipt, the server determines if the target nickname (*nickname2*) exists on the server. If not, an error is issued, otherwise, it first initializes both clients by relaying: *320 MESSAGE +create nickname1_to_nickname2_unixtime.meml*. Again, unixtime is used to distinguish chat sessions between these two clients, as well as to synchronize the filename on both clients.
- This message prompts the clients to creates a file locally (i.e. nickname1_to_nickname2_1067338605.meml). The clients then display this file on their web browsers.
- After file creation, the initiating client can update its dialogue file with the sent message immediately.

- The server then sends the initial message to the receiving client: *350 MESSAGE nickname1 nickname2 Hello nickname2!*

- Upon receipt, *nickname2* updates its local dialogue file with the message.

- At this point, the initialization phase is complete, and all subsequent messages are sent to the target client directly through the server while assuming that the source client updates its copy upon the message send.

- Another difference from classroom dialogues is that the conversation can end at any time without further processing (i.e. no PART message is needed).

The reason for the # prefix in classroom naming now becomes apparent. Because nicknames are arbitrary, a user could name him/herself Math101. The # character identifies the difference between these entities.

6.8. Disconnecting from Server

Disconnecting from the server itself is done in similar fashion to the PART command. Again, we will assume that our client, *nickname2*, wanted to disconnect from the server. For arbitrary cases, let us also assume that *nickname2* was an active participant in the following classrooms: #Math101, #Algebra102, #Geometry204.

- *Nickname2* sends *QUIT good bye!* to the server.

- This command is always granted by the server. The server sends: *317 QUIT +ok*.

- Upon receipt of reply 317, the client breaks the connection with the server.

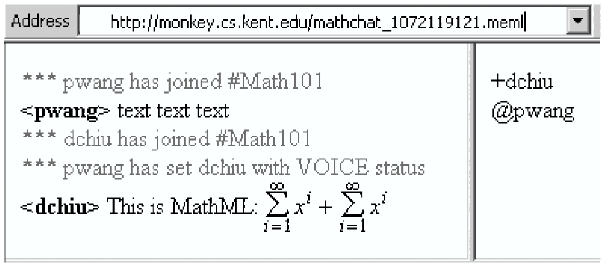
- The server now determines which classrooms *nickname2* is associated with (#Math101, #Algebra102, #Geometry204), and updates the respective classroom dialogue files with the QUIT message and transmits this message to all participants in the same classrooms with: *316 QUIT nickname2 good bye!*.

- Upon receipt of reply 316, clients update their dialogue file with the QUIT message.

- The QUIT command is complete and *nickname2* no longer exists on the server.

Figure 2 shows a typical classroom scenario from the perspective of *pwang*.

The following is the MeML source code for the dialogue window in respect to Figure 2.



http://monkey.cs.kent.edu/mathchat_1072119121.meml

*** pwang has joined #Math101
<**pwang**> text text text
*** dchiu has joined #Math101
*** pwang has set dchiu with VOICE status
<**dchiu**> This is MathML: $\sum_{i=1}^{\infty} x^i + \sum_{i=1}^{\infty} x^i$

+dchiu
@pwang

Figure 2. A MathChat Classroom Scenario

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
 href="http://www.w3.org/Math/XSL/
 mathml.xsl"?>
<meml>
<html:head>
 <html:meta http-equiv="Content-Type"
  content="text/html; charset=iso-
  8859-1" />
</html:head>
<html:body>
<html:b>***</html:b> pwang has joined
        \#Math101<html:br />
<html:b>&lt;pwang&gt;</html:b> text
        text text<html:br />
<html:b>***</html:b> dchiu has joined
        \#Math101<html:br />
<html:b>***</html:b> pwang has set
        dchiu with VOICE status
</html:font><html:br />
<html:b>&lt;dchiu&gt;</html:b>
This is MathML:
<m:math>
 <m:mrow>
  <m:mstyle displaystyle="true">
  <m:munderover>
   <m:mo>&sum;</m:mo>
   <m:mrow><m:mi>i</m:mi><m:mo>=</m:mo>
    <m:mn>1</m:mn>
   </m:mrow>
   <m:mi>&infty;</m:mi>
  </m:munderover>
  <m:msup><m:mi>x</m:mi><m:mi>i</m:mi>
  </m:msup></m:mstyle>
 <m:mo>+</m:mo>
 <m:mstyle displaystyle="true">
 <m:munderover>
  <m:mo>&sum;</m:mo>
  <m:mrow><m:mi>i</m:mi><m:mo>=</m:mo>
   <m:mn>1</m:mn>
  </m:mrow>
  <m:mi>&infty;</m:mi>
 </m:munderover>
 <m:msup><m:mi>x</m:mi><m:mi>i</m:mi>
 </m:msup></m:mstyle>
 </m:mrow>
</m:math>
</html:body>
</meml>
```

From [18], the MeML processor sits on the client's machine as a plug-in to their web browsers. Because our foundation for handling chat dialogue files is an MeML file, all MeML-element processing is done on the client side. This includes support for computations, mathematics expression display, etc. This also leaves the client with flexibility to apply style to this content-oriented file.

7. Future Work

The entirety of MathChat is currently being specified. We delve into its practical uses in order to refine its functionalities. Currently, MathChat has a many-to-one relationship with clients to servers. As server loads increase, an incorporation of multiple servers may be of use. A specification of a server-to-server MathChat protocol is necessary to provide for this need.

It has also come to our attention that mathematical expressions may be tedious to type out with MathML constructs. While MathChat's predecessor, WMEchat supported $\LaTeX$ input, we have yet to analyze the support for $\LaTeX$ in MathChat. A server-side $\LaTeX$ to MathML converter may offer us this ability.

8. Acknowledgments

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

I would like to express my deepest gratitude to my colleagues at the Institute for Computational Mathematics (ICM/Kent), and especially to my advisor, Professor Paul S. Wang for direction, guidance, and support for this work.

References

[1] Blackboard. www.blackboard.com.
[2] Extensible hypertext markup language. www.w3c.org/MarkUp.
[3] Maple. www.maplesoft.com.
[4] Mathematical markup language. www.w3c.org/Math.
[5] Matlab. www.mathworks.com.
[6] Php: Php hypertext processor. www.php.net.
[7] Rfc: The internet relay chat protocol. www.irchelp.org/irchelp/rfc.
[8] Topclass. www.wbtsystems.com/products/lms.
[9] Webct. www.webct.com.
[10] Florida virtual school: The future of learning? Technical report, October 2002. www.aypf.org/forumbriefs/2002/fb101802.htm.
[11] D. Chiu, Y. Zhou, X. Zou, and P. S. Wang. Design, implementation, and processing support of meml. *Proceedings of Internet Accessible Mathematical Computation 2003 Workshop*, July 2003.
[12] J. Johnson, Y. N. Lakshman, T. T. Hewett, T. Souder, T. Fitzgerald, S. Donegan, and P. Morgovsky. Virtual office hours using techtalk, a web-based mathematical collaboration tool. In *Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education*, pages 130–133. ACM Press, 1998.
[13] W. Liao and P. S. Wang. Specification of omei v1.0: Open mathematical engine interface.
[14] W. Liao, Y. Zhou, X. Zou, and P. S. Wang. Open mathematical engine interface and its application. *Submitted to Journal of Symbolic Computation*.
[15] M. Simonson, S. Smaldino, M. Albright, and S. Zvacek. *Teaching and Learning at a Distance: Foundations of Distance Education (2nd ed)*. Merrill/Printice Hall, Upper Saddle River, NJ, 2003.
[16] P. S. Wang, , Y. Zhou, and X. Zou. Mathematics education markup language. *Proceedings of E-Learn 2002 World Conference on E-Learning in Corporate, Government, Healthcare, & Higher Education*, October 2002.
[17] P. S. Wang, N. Kajler, Y. Zhou, and X. Zou. Initial design of a web-based mathematics education framework. *Proceedings of Internet Accessible Mathematical Computation 2002 Workshop*, June 2002.
[18] P. S. Wang, N. Kajler, Y. Zhou, and X. Zou. Wme: Towards a web for mathematics education. *Proceedings of ISSAC 2003*, pages 258–265, August 2003.
[19] P. S. Wang, Y. Zhou, and X. Zou. Web-based mathematics education: Meml design and implementation. *Proceedings ITCC'04, IEEE*, April 2004.
[20] X. Zou. Xmec: An extensible mathematical encoding converter. Technical report.