Towards Global Collaboration Tools

Eiman Elnahrawy[*]
Department of Computer Science
Rutgers University
Piscataway, NJ 08854, USA
eiman@paul.rutgers.edu

William C. Cheng
Department of Computer Science
University of Southern California
Los Angeles, CA 90089, USA
bill.cheng@acm.org

Leana Golubchik[†]
Department of Computer Science, IMSC, and ISI
University of Southern California
Los Angeles, CA 90089, USA
leana@cs.usc.edu

ABSTRACT

The introduction of collaboration applications on the Internet has enriched communication between diverse users via chatting, and audio and video conferencing. It has also facilitated numerous real life tasks such as conducting of business meetings on the web, distance learning, and distributed authoring. Unfortunately, there are several difficulties that prevent collaboration applications from being ubiquitous on the Internet. The major difficulty is the lack of standards for communication between different collaboration applications which restricts collaboration to only instances of the same implementation, and in turn, limits the usability of such applications. In this paper, we propose a novel collaboration protocol for the exchange of information between users in collaborative environments. The proposed protocol is not tied to a specific implementation of collaboration tools which would enable several tools of the same type to communicate with each other independently of their implementation details. We also discuss a framework architecture for integration of several collaboration tools into one application. The proposed framework utilizes our proposed protocol, and adopts an HTTP-based communication model in order to overcome another difficulty which is possible firewalls constraints. In particular, we extend the standard HTTP streaming technology to accommodate several forms of collaboration.

Keywords

Collaboration Tools, Usability, Firewalls, HTTP Streaming

1. INTRODUCTION

Over the past few years the Internet has grown in popularity and in capabilities. Numerous new applications that utilize the Internet have emerged. Currently, the Internet is used not only for storing and exchanging of information, but also for several forms of collaboration between heterogeneous users. In general, there are three forms of collaboration between users on the Internet [1]: communication, coordination, and production. Communication refers to the situation where users can just communicate with each other by exchanging e-mails, discussing topics in chat rooms, and so on. Coordination refers to the situation where users coordinate to reach some conclusions such as voting for the best time to schedule a meeting. Production refers to the situation where the aim of the collaboration is to produce materials such as collaborating on a business report to produce its final version, or collaboration between students to produce a project report, and so on. In general, collaboration tools are divided into two categories: synchronous and asynchronous tools. Asynchronous collaboration tools refer to the set of tools that do not necessarily require online synchronous interaction between users. They include, but are not limited to, Bulletin Board systems, Usenet, e-mail, mailing lists, document sharing via the well known ftp protocol, off-line auctions, and distributed authoring tools. This category of collaboration tools is not the focus of this paper and we mentioned it for completeness of description only. In what follows we use the word collaboration tools to refer to synchronous collaboration tools. Synchronous collaboration tools are those that require online interaction between users. They have evolved with the growth of the Internet from simple text based conferencing (chatting) and multiple user dungeons (MUDs) to more sophisticated audio and video conferencing, as well as web presentations, whiteborads, voting tools, interactive games, etc. Individual collaboration tools, as well as collaboration applications that combine several tools in one framework, are currently widely available on the Internet, e.g., (a) ICQ [2] and mIRC [3] for online chatting and instant messages, (b) iSpQ [4], CU-SeeMe [5], and Microsoft NetMeeting [6] for audio and video conferencing, (c) peer-to-peer applications, etc. These applications have facilitated numerous real life tasks such as distance learning, conducting of business meetings on the Internet, collaborative medical diagnosis, distributed authoring, and so on.

[*]This work was done while the author was at the Department of Computer Science at the University of Maryland, College Park.

[†]This work was partly done while William Cheng and Leana Golubchik were with the Department of Computer Science and UMIACS at the University of Maryland, College Park. This work was supported in part by the NSF ANI-0070016 grant.

1.1 Motivations and Approach

With the exception of a few chat tool implementations that use the standard IRC protocol [7], the major objective of the current collaboration frameworks such as [1, 8] is to define standards for the overall communication between their applications, which would enable the integration of several collaboration tools in one application. However, they offer no global standard for the specific exchange of information between the corresponding tools. Even simple individual tools such as instant messages have no global standard for the exchange of information. This generally restricts the collaboration to only instances of the same application. The lack of communication standards between different implementations has limited the usability of such applications since users with different implementations cannot collaborate with each other. In general, this difficulty has prevented collaboration applications from being ubiquitous on the Internet. Users from different, possibly diverse, communities do not necessarily have access to the same collaboration tool implementation. Limiting collaboration to instances of the same implementation directly violates the major objective of collaboration, i.e., bringing diverse users together. Although such standardization across different implementation seems obvious, it is surprising that there is not any effort to define and specify some standard collaboration protocols. This is most likely due to the fact that some of the market-dominating interactive tools, e.g., AOL and MSN, may not want to support such inter-operability, and may even strive to prevent it in order to preserve their own customer bases. However, we argue that this is not fair for the community since it prevents users from diverse countries as well as users from the United States from enriching their cultural communication. We argue that when collaboration protocols become real standards, these companies will have to conform. Therefore, our first contribution is to introduce a novel protocol for the exchange of information between collaboration tools. This protocol is generic, i.e., it is not tied to a specific implementation of collaboration tools that would enable collaboration tools of the same type, but not necessarily of the same implementation, to communicate with each other. The protocol is based on exchanging the minimum amount of information about any update in the collaborative environment between the users. We hope that our proposed solution becomes the first step toward supporting inter-operability in collaboration environments.

Our second contribution in this paper is the design of a general framework architecture for collaboration environments that integrates several tools and utilizes our proposed protocol. The framework adopts an HTTP-based communication model in order to overcome another difficulty which is possible firewalls constraints. In general, firewalls have emerged on the Internet due to the growth of malicious intrusions. They safeguard both personal computers and corporate networks, for provision of privacy and integrity of personal information. Unfortunately, firewalls and collaboration applications have conflicting goals since existing collaborative environment frameworks are based on a heavy use of TCP and UDP ports for exchanging information [5, 1, 6, 8]. These ports are likely to be blocked by firewall administrators since they are considered potential threats that carry viruses or possible attacks [9]. This problem is not trivial since there are several administrative domains at each user's side; which results in more complicated constraints that vary from user to user. It is important to point out that some classes of tools such as peer-to-peer collaboration tools systems have addressed firewalls problems. They may even use HTTP ports as we do, however, our solution is different in that it uses the standard HTTP protocol itself and not just abusing HTTP by tunnelling through its port. In particular, we extend the standard HTTP streaming technology for streaming video to accommodate all forms of collaboration. Although this HTTP-based communication serves as another step toward making collaboration applications more usable, our proposed collaboration protocol is not coupled with any specific form of communication, and other models such as communication using TCP ports can also be used.

Our last contribution is a prototype implementation for our proposed framework and protocol. The prototype has been developed entirely using the Java programming language [10]. The prototype is still preliminary; however, the major functionality is mature enough which encouraged us to present it. In general, we aim at a proof of concept, i.e., a standard protocol and support of users behind firewalls using the standard HTTP streaming technology.

1.2 Organization

The rest of this paper is organized as follows. Section 2 gives a brief description of our proposed architecture of collaboration applications that support several tools, as well as our proposed HTTP-based communication model. An overview of our proposed collaboration protocol is presented in Section 3. Section 4 discusses the specifications of the protocol. An overview and a demonstration of the prototype implementation are presented in Section 5. Section 6 discusses related work and existing collaboration applications. Finally, Section 7 concludes this paper.

2. FRAMEWORK ARCHITECTURE

This section discusses our proposed framework architecture and gives an overview of our proposed HTTP-based communication model. In general our architecture is based on the client-server model. We assume that the clients can communicate with the collaboration server(s) using *messages*. We adopt an HTTP-based communication model in order to overcome any existing firewalls constraints. However, we do not claim that this is the best way to communicate in collaboration environments nor do we claim that all functionality provided in similar systems such as [1] can *efficiently* be supported in this model. In general, some classes of collaboration tools such as peer-to-peer systems have tunnelled through HTTP ports in order to bypass firewalls. Our approach, on the other hand, is completely different. We do not abuse HTTP by tunnelling through HTTP port 80, rather, we extend the standard HTTP streaming technology of the HTTP protocol in order to accommodate all forms of collaboration. It is important to notice that any communication model, e.g., one based on using TCP ports, can also be used. Moreover, gateways that support cross-communication models is another interesting extension to our current system. We leave these issues to future work.

2.1 Communication Model

The general form of communication between a client and a web server on the Internet is via individual HTTP request-response. In this form of communication, the client web browser sends an HTTP request to the web server requesting some data, then the server sends an appropriate response to the client, and then the connection is closed. There is no further interaction between the client and the server unless the client initiates another HTTP request to the server. In contrast, HTTP streaming technology has made this further interaction possible. In the HTTP streaming technology, also known as "server push", the web server includes information in the response header, informing the client web browser to expect receiving more data from the server. The connection between the server and the client remains open until either the server sends an end marker to the client or the client interrupts the connection. The server pushes blocks of data to the client down the connection whenever it wants to. HTTP Streaming technology utilizes the MIME message formats by using a variant of the "multipart/mixed" content type called "multipart/x-mixed-replace". It has been originally introduced to support animation in the Netscape Navigator web browser. Currently, it is used in a limited number of applications such as streaming of continuous media and transmitting live images, captured by webcams, to the web browser.

We extend the HTTP streaming technology as follows. We assume HTTP/HTTPS collaboration servers where all communication between the clients and the server is performed via HTTP. The data blocks routed in the collaboration environment are encoded as messages. This message passing enables all the users to maintain the same state, i.e., share all the updates. The central server uses "HTTP streaming" to *push* any new message to every client in the environment while the clients communicate with the server using "individual HTTP Post requests". Each individual post request corresponds to one message, and the content-type header field of the request corresponds to the type of the message. There are two types of messages used in the communication between the client application instances and the central server: control messages and event messages. The control messages are used for controlling the environment; for example, when new users wish to join the environment, their clients exchange control messages with the server. The event messages carry information about any update in the collaboration tools; for example, a new chat message sent by one of the users is delivered to the central server to be then routed to the other users. The specification of the message format will be discussed later in Section 4. Experimental MIME content types [11, 12] are used to define a content-type for each type of the messages, e.g., an "application/x-event-chat" type is used for an event message that carries an update in a chat tool, while an "application/x-control-join" type is used for a join-session control message. The content-type information is used by the client application and the central server to interpret the messages. This information also allows integrating several collaboration tools into one client application. Standard structures for the data encapsulated in any message is also defined based on the content-type of the message as we will show in Section 4. These standards ensure inter-operability of collaboration tool implementations. The central server opens an HTTP stream to every user in the environment once they successfully join the environment. It then uses these streams to push new messages to the users down the stream. We introduce a variant of the "multipart/mixed" content type called "multipart/x-mixed-update" to open a stream from the central server to the clients. The intuition behind this type is that the routed messages carries updates in the tools, i.e., the current view of the tool is only updated and not entirely replaced by the new information as in the former "x-mixed-replace" type used for streaming new video frames. It is worth mentioning that the defined standards for the *type* and the *structure* of messages provide protection against hackers that may take advantage of our streaming model. In particular, the server and/or the client implementation can discard any suspicious message that does not conform to these standards.

2.2 Architecture Elements

Figure 1 shows the overall architecture. The architecture consists of two major elements: the central server(s) and the client application. The client application can be further divided into two elements: the core element and the local collaboration tools. The dashed line represents the communication between the client and the server using individual HTTP requests. The solid line represents an HTTP stream, used by the server to communicate with the client. As we mentioned above, these two lines can be replaced by any other communication model. The overall communication model may contain one central server or multiple distributed servers connected together. In case of multiple servers, the servers should be capable of exchanging information with each other in order to maintain consistency and integrity of the collaborative environment (i.e., exchange information about participating users, messages, etc.). This functionality is not yet supported in our current prototype. We believe that the current implementation can be fairly extended to this multi-server case similar to Web servers, however, we leave this extension as well as a study of the scalability and availability issues to future work. In both cases, our view of collaboration servers is that they will be deployed in the public domain as a provided service similar to the IRC protocol [7]. In the rest of this section we briefly discuss the functionality of each element in the architecture. In general, the functionality of collaboration environments has been studied and defined in different earlier systems such as [1, 8]. The major functionality of any collaboration environment consists of session management, e.g., management of participating users, communication management, access control via authentication, privacy management, and activities logging for supporting temporary disconnection, monitoring [13], etc. Our system also supports this functionality with minor differences. Due to limited space we omit the details of this part.

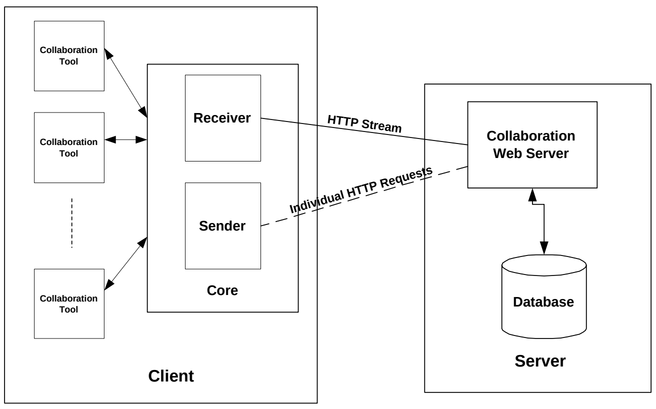


Figure 1: Overall architecture of collaboration environments

2.2.1 *Central Server*

The central server is responsible for: (1) storing all the information about the collaborative environment in a database. The information may include the supported collaboration tools, the current participants, and their modes, e.g., active or away, and so on, (2) routing event messages between the clients in the environment distributing the event message, received from a client, to all participating clients, (3) maintaining the state of the environment by exchanging control messages with the clients for authentication, granting privileges, leaving the environment, etc., and finally, (4) logging all the activities in the environment.

2.2.2 *Client Application*

The client application is divided into two elements, the core element and the local collaboration tools. The core element is generally responsible for handling the two way communication between the client implementation and the server. In general, it is responsible for processing all outgoing client-to-server and all incoming server-to-client control messages. It is also responsible for delivering all outgoing client-to-server event messages from the collaboration tools to the central server, and for delivering all incoming server-to-client event messages from the central server to the appropriate tool based on the message content-type. The specific tasks of the core element can be summarized as follows: (1) enabling the user to join the collaborative environment by exchanging all the necessary information with the central server about the user name, password, etc., (2) managing the two way communication between the client application and the central server after joining the environment, (3) performing the appropriate action(s) upon receiving any message from the central server based on the type of the received message, etc., and finally, (4) privacy management by encrypting any event message before sending it to the central server and decrypting all the messages received from the central server before delivering the message to the appropriate collaboration tool. The local tools are responsible for: (1) parsing any messages delivered by the core element to extract all the necessary information about the tool update, (2) performing the appropriate action based on the extracted update information, (3) encapsulating any update in the tool, performed by the user, in an event message, and then delivering the message to the core element, in order to be encrypted and sent to the server.

3. COLLABORATION PROTOCOL

This section gives an overview of our proposed collaboration protocol. In general, the collaboration protocol is event-driven. It is based on exchanging the minimum amount of information about any update in the environment, between the clients, in the form of messages. As we defined above, there are two types of messages used in the communication between the client application instances and the central server: control messages and event messages. Each client extracts the information included in these routed messages and interprets it in order to perform the necessary updates. This collaboration protocol provides standards for communication between individual synchronous collaboration tools such as a chat tool, a web presentation tool, a whiteboard tool, a file transfer tool, audio and video tools, a voting tool, etc. In addition, it also supports applications that integrate several tools into one collaboration application since the routed event messages are distinguished via a unique type for each tool. The major advantages of this protocol are flexibility and efficiency with respect to the amount of information routed in the collaborative environment. The flexibility is due to its capability to accommodate several types of collaboration tools independently of their implementation details, while the efficiency is due to the fact that only the necessary information about the updates is exchanged in the collaborative environment.

4. SPECIFICATIONS

We discuss the specification details of our proposed collaboration protocol in this section. This protocol is still in its early version, i.e., we study adding and removing features from the current specifications. Our goal is to support the ultimate generic functionality of each collaboration tool. We will demonstrate each type of messages by several examples as the space permits, for complete description of the protocol specifications please refer to [14].

4.1 Control Messages

Control messages are used for controlling the environment, e.g., a join-session message allows new users to join the environment, while a users-list message allows registered users to receive a list of the current participating users. Other control messages include, but are not limited to, new-user message and leave-session message, to inform current registered users that a participant has joined or left the environment, respectively, grant-privilege message to grant the moderator privilege to a specific participant, etc. Client-to-server control messages are processed by the central server. The central server performs the appropriate action(s), such as updating the session information, sending server to client(s) messages, or both, depending on the type of the control message. On the other hand, server-to-client messages are processed by the client application instance. The client application performs the appropriate action such as updating the users' list, depending on the type of the message.

4.1.1 Examples

As an example of the format of control messages, consider a join-session message sent from user "A" to the central server to join the environment. The content-type of the message is "application/x-control-join", while the content of the message includes the user name, nickname, session type (public = 0 and private = 1), password in case of private sessions, etc., in a specific order, and separated by the appropriate separators. Another example is changing the mode of one participating user. If user "A", with nickname "johny", wants to change its mode from active to away (active = 1 and away = 2) it sends a client-to-server change-mode message to the central server. The central server then notifies all participating users using a server-to-client change-mode message with content-type "application/x-control-mode". The content of the message in this case is the nickname of the user who changed the mode, and the new mode of that user, separated by the appropriate separators, e.g., `johny|2`.

4.2 Event Messages

Event messages do not play any role in controlling the environment. They are used for distributing updates in the collaborative environment to all participating users. They contain two kinds of information: the content of the message and the content-type of the message. The content of the message is the description of the tool update. The content-type of the message determines which tool should perform that update. Event messages are initiated by the client applications and not by the central server. In contrast to control messages, the central server does not perform any processing of the content of event messages. It is only responsible for routing these messages, using the same content and content-type of the message, between the participating users. By default the messages are routed to every user. However, users may specify who should receive their messages, e.g., a private chat message in a chat tool can be addressed to a specific recipient. This is achieved in our proposed HTTP-based communication model by including a parameter in the client-to-server HTTP requests that includes the recipient's nickname. The central server then pushes the message to the specified user only. When the client application receives an event message from the central server, it is responsible for performing the appropriate update to the corresponding tool, depending on the content-type of the message.

4.2.1 Examples

As an example of the format of event messages, consider a shared web browser tool. When user "A" with nickname "amy" clicks on a hyperlink or loads a specific page, an event message is sent to the other participating users. The content type of the message is "application/x-event-hyperlink", while the content of the message includes the user's nickname, and the hypelink, in a specific order, separated by appropriate separators, e.g.,

```
amy|http://www.yahoo.com
```

Similarly, if the same user sends a chat message to the other users, then the content type of the message in this case is "application/x-event-chat", while the content of the message includes the font name, font size, font color, font bold style (true or false), font italic style (true or false), nickname of the sender, and the actual text of the chat message, e.g., `Serif|12|-65536|F|T|amy|Hello`. It is worth mentioning that we handle video and audio streaming in a similar way. For example, we stream video as a sequence of captured live images in the form of gif or jpg frames every predefined amount of time, based on the desired frame rate. In this case, the event message content includes the user's nickname, the file type (gif or jpg), the major and minor sequence numbers, and the frame bytes, e.g., `jimmy|gif|100|12<CR-LF>video bytes`
Where `<CR-LF>` refers to a carriage return followed by a line feed, while the content of the audio event message includes the user's nickname, the audio frame information (the major and minor sequence numbers, encoding, rate, etc.), and the audio bytes. The content of a file transfer event message is the nickname of the sender followed by the file full name (name.extension), file size in bytes, separated by appropriate separators, then a `<CR-LF>` pair

followed by the file bytes, e.g., to transfer the “abcd.pdf” file, user “amy” sends an event message with the following content.

```
amy|abcd.pdf|1345<CR-LF>file bytes
```

5. PROTOTYPE IMPLEMENTATION

We have built a prototype implementation of our proposed architecture entirely in the Java programming language [10]. The implemented architecture utilizes our proposed protocol and uses our HTTP-based communication model. In particular, the client and the server sides were developed using Java 2 Software Development Kit JDK1.3.1. [10]. Java Web Server JSDK 2.1 offered by Sun Microsystems as a stand-alone servlet engine [10] was used for running the server prototype. Other stand-alone or add-on servlet engines offered by several other companies can also be used for running the server side implementation [15].

5.1 Server Side

We used a Java servlet [16] for the server side implementation in order to utilize two basic features of servlets in our prototype: (1) support of several concurrent users efficiently, (2) ability of interacting closely with the web server in order to access other services on the server, e.g., to access a database residing on the server. In addition, servlets are written Java which is both object-oriented and platform independent. The current prototype supports the basic functionality of the server side. The servlet is able to process any event message and log all the activities in the session. It can also process the major control messages. All the synchronization and concurrency are functioning properly. The server implementation is capable of properly handling concurrent users, and successfully routing messages in the session.

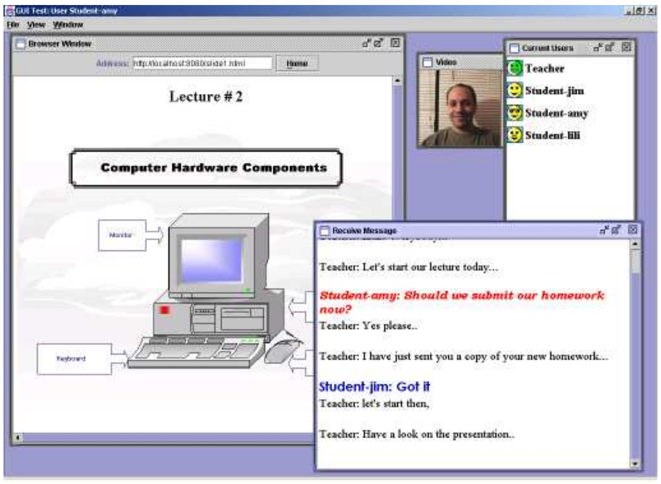
5.2 Client Side

We also used Java for the client side implementation due to its portability (i.e., more usable) and multi-threading features [10, 16]. A stand-alone Java application was chosen over a Java applet since Java applets are not persistent, i.e., they must be downloaded, every time, before execution. Nevertheless, Java applets cause several security-related problems on the client side due to the lack of trust in remotely downloaded code. A set of collaboration tools have been implemented and integrated into our client prototype. The current prototype contains chat, file transfer, audio, video, and web browser tools. The basic functionality, discussed in Section 2, of the core element as well as of each collaboration tool is supported in our current prototype. More sophisticated tools, such as a witeboard tool, can be implemented and integrated into our prototype in the same way.

5.3 Demonstration

To demonstrate our prototype, consider a typical distance learning environment. The teacher presents the material to the students using a presentation tool such as a web presentation tool. The participants may discuss some points of the presentation. The teacher may narrate, show some video clips or live images to the students. The students may submit their homework using appropriate tools, etc. Our prototype supports this scenario by offering the following tools: (a) a chat tool for discussion between participants, (b) a web browser for material presentation, (c) an audio tool for narration, (d) a video tool for showing the teacher, and (e) a file transfer tool for submitting homeworks, reports, etc. Each client implementation also displays a list of the current participating users. Figure 2 shows a snapshot of this scenario using our prototype.

5.4 Discussion



Teacher: Let's start our lecture today...

Student-amy: Should we submit our homework now?

Teacher: Yes please..

Teacher: I have just sent you a copy of your new homework...

Student-jim: Got it

Teacher: let's start then,

Teacher: Have a look on the presentation..

Figure 2: Distance Learning Session: Student Application.

It is important to distinguish between our proposed approach and the implemented prototype at this point, since the major objective of the current prototype is to demonstrate the soundness and the applicability of our approach. Several extensions to the current prototype are still needed. These extensions are part of our future work. For example, we argue that the proposed protocol support communication of tools of the same type independently of their implementation details. We realize that this argument need to be strengthened by demonstrating an example using different implementations of the same tool that are built using our proposed protocol. Also, supporting plugins that work across existing tools that do not support our protocol, e.g., plugins that enable ICQ and MSN to communicate. Since these tools do not utilize our protocol, the plugins work as proxies at each user’s side to communicate with the server, using the proposed protocol, on behalf of the implemented tool. Another extension is to compare the performance of other techniques that can be used in implementing the server and the client sides. We favored Java over other techniques in our implementation. However, there are other techniques that can be used such as .NET, CORBA, or even Web services. Nevertheless, making the interface more usable is an important issue that we plan to investigate.

Finally, designers wish to use our proposed approach can either use our existing implementation of the “core” element, discussed in Section 2, and interface their implemented tool(s) using the currently provided Java API, or they can use our protocol to communicate with the implemented server and implement their own collaboration application. If they choose the first way, any tool only needs to know how to interact with the Sender and the Receiver parts of the client core element using the provided API. If they choose the second way, we urge them to also utilize our proposed architecture since the communication between the client and the server in our architecture is transparent to all the tools. As we briefly mentioned above, we expect that collaboration servers will be widely deployed in the public domain similar to the IRC servers, and that any collaboration tool/application that conforms to our protocol will be able to communicate with these servers in order to collaborate with other participating users. For more details please refer to [14].

6. RELATED WORK

Several academic prototypes, such as Habanero [1] and Tango [8], have been developed in order to support collaboration between users. Their goal is to integrate several collaboration tools into one multi-user collaboration application. These prototypes are built using sophisticated frameworks where the interaction and the integration between several tools are well defined. However, in contrast to our proposed collaboration protocol, they offer no standard for the specific exchange of information between the corresponding tools. This clearly restricts the collaboration to only tools of the same implementation, e.g., a chat tool built in Habanero cannot exchange information with a chat tool built in Tango. Another key difference between our approach and these prototypes is support for firewalls. These systems are usually based on the client-server model and the communication between the central server and the clients is performed via TCP and UDP ports. These ports are likely to be blocked by firewall administrators since they are considered potential threats that carry viruses or possible attacks [9]. Our approach is also based on the client-server model. TCP/UDP-based communication can also be used, however, we also propose an HTTP-based communication model in order to solve possible firewalls conflict. Numerous commercial products for collaboration applications are also widely available, e.g., iSpQ video chat [4], Microsoft NetMeeting [6], and simple chat applications such as mIRC [3] and ICQ [2]. These products, unlike our proposed architecture, are considered primitive since they support limited number and type of collaboration tools. They also do not usually conform to standards, and hence the only form of collaboration is via identical instances of the application. In addition, developers of such commercial products do not traditionally take into consideration possible security constraints during development of their collaboration applications. Consequently, their applications usually have certain inflexible requirements that must be satisfied in order to work properly from behind firewalls [17, 5, 1, 6, 8]. Several standard protocols have been introduced in order to support certain forms of collaboration between users. For example, the Internet Relay Chat (IRC) [7], and the H.323 (IP) protocol [18]. These protocols, compared to our approach, support only one form of collaboration, that is text conferencing, and audio and video conferencing, respectively. Moreover, applications built using these protocols will not run from behind firewalls, since these protocols use TCP and/or UDP ports in their communication. The last class of collaboration applications that is relevant to our work is the class of peer-to-peer systems such as Groove [19]. Applications of this class also do not address our proposed standardization approach. They either have certain requirements in order to work properly behind firewalls, e.g., opening of some TCP ports, or use HTTP tunnelling which is basically a relay of communication via the HTTP port. Their solution for overcoming firewalls is completely different from our proposed communication model. Our model is not based on tunnelling through HTTP port. We consider HTTP tunnelling an abuse of the HTTP protocol. Rather, our approach is based on extending the standard HTTP streaming technology to accommodate collaboration. Finally, we presented a short poster on this work in [20].

7. CONCLUSIONS

We addressed several difficulties that prevent collaboration applications from being ubiquitous on the Internet and deprive several communities of users from utilizing such useful technology. In particular, we proposed a solution for two specific problems. The first is the lack of standards for communication between different collaboration applications which restricts collaboration to only instances of the same implementation. For this problem, we presented a novel protocol for the exchange of information between users in a collaboration environments and discussed its advantages over existing frameworks. We also proposed an HTTP-based communication model for collaboration in order to overcome a second problem which is the conflict between firewalls and collaboration applications. We discussed the overall architecture of our system. The system utilizes our proposed collaboration protocol and adopts our HTTP-based communication model. It enables integration of several collaboration tools into one application framework. We briefly described the functionality of the architecture. Finally, we discussed the implementation of our current prototype. The system, in general, is still in its early version. We basically aimed at a proof of concept and tried to highlight major future work directions.

8. REFERENCES

[1] A. Chabert et al., “NCSA Habanero - synchronous collaborative framework and environment (white paper), http://havefun.ncsa.uiuc.edu/habanero/whitepapers/ecscw-habanero.html.”

[2] “ICQ Inc., the ICQ Internet Chat Service home page, http://www.icq.com.”

[3] “Tjerk Vonck and mIRC Corporation, the mIRC home page, http://www.mirc.com.”

[4] “nanoCom Corporation, the iSpQ home page, http://www.ispq.com.”

[5] “Cornell University’s CU-SeeMe web page, http://www.imj.org.il/shazan/cornell.html.”

[6] “Microsoft Corporation, Windows NetMeeting home page, http://www.microsoft.com/windows/netmeeting/.”

[7] J. Oikarinen, “Internet Relay Chat RFC.” RFC1459, 1993.

[8] L. Beca et al., “Tango - a collaborative environment for the world-wide web, http://citeseer.nj.nec.com/132092.html,” tech. rep., Syracuse University, 1997.

[9] D. B. Chapman and E. D. Zwicky, *Building Internet Firewalls*. O’Reilly, first ed., September 1995.

[10] “Sun Microsystems Inc., the java.sun.com home page, http://www.java.sun.com.”

[11] N. Borenstein and N. Freed, “Multipurpose Internet Mail Extensions (MIME) Part One.” RFC2045, 1996.

[12] N. Borenstein and N. Freed, “Multipurpose Internet Mail Extensions (MIME) Part Two.” RFC2046, 1996.

[13] E. Elnahrawy, “Log-based chat room monitoring using text categorization: A comparative study,” in *The IASTED International Conference on Information and Knowledge Sharing (IKS 2002)*, November 2002.

[14] E. Elnahrawy, “An HTTP-Based Distributed Application Framework For Interactive Group Collaborative Environments,” Master’s thesis, University of Maryland, College Park, MD 20742, USA, May 2002.

[15] “Sun Microsystems Inc., Java Servlet Technology: The power behind the server, http://java.sun.com/products/servlet/.”

[16] B. Eckel, *Thinking in Java*. Prentice-Hall, second ed., 2000.

[17] “CollabWorx Inc., Integrated Web Collaboration Solutions, http://www.webwisdom.com/.”

[18] “Video Development Initiative, Video Conferencing Cookbook,” April 2000. http://www.vide.gatech.edu/cookbook2.0/.

[19] “Groove Networks, Inc., http://www.groove.net/.”

[20] E. Elnahrawy, W. Cheng, and L. Golubchik, “HTTP-ICE: An HTTP-based distributed application framework for Interactive Collaborative Environments (poster),” in *The 12th International WWW Conference*, (Budapest, Hungary), 2003.