

SOA-based Collaborative Multimedia Authoring

Andrew Roczniak*, Salinah Janmohamed*, Christian Roch*, Abdulmotaleb El Saddik* and Pierre Lévy†

*Multimedia Communications Research Laboratory

†Collective Intelligence Laboratory

University of Ottawa, Ottawa, Canada

Email: {roczniak, abed}@mcrmlab.uottawa.ca, plevy@uottawa.ca

Abstract—The pace of technological change requires applications to constantly evolve. Service Oriented Architecture (SOA) promises to render this evolution more flexible. We are developing an application that allows collaborative authoring and manipulation of multimedia documents. This application is defined in terms of services, and partly based on the Jabber set of protocols. We measure the performance of the application when used over the Internet and using a public Jabber server.

I. INTRODUCTION

Service Oriented Architecture (SOA) promises greater adaptability of business processes to new and evolving business realities. Conceptually, we can represent it as applications being built on top of services (and not frameworks), or alternatively as reversing the relationship between services and frameworks - services are now built on top of frameworks - allowing for specialized and highly-tuned services to do the work required by an application. The concepts of functionality and service in SOA are closely related. Although what one can consider *functionality* to be is virtually unlimited, what is a *service* depends how a specific functionality is packaged and offered to anyone interested in that functionality.

Services are self-contained and operate as black boxes; all necessary components to accomplish a specific functionality are encapsulated within the service and consumers neither know nor care how they perform their function, as long as they return the expected result. Those characteristics should make a service independent of consumer context, allowing reuse in contexts not known at design time. Services expose their functionality through interfaces. The requirement on those interfaces is that they be *invokable* in a standardized way. The service itself may be within the same application or in a completely different system on the Internet but the interconnect scheme or protocol used to allow the invocation should however be open and standardized. This in turn requires open and well-understood way of publishing, finding and binding to services. The infrastructure components required to make the connection can be diverse since services may be implemented on a single machine or distributed across a set of computers on a local area network [7]. Services can be composed into other services - based on quantifiable quality of service, for example - and offered transparently to the consumer, as one service. Exposing functionality as services allows greater flexibility in application design and integration. The application as a whole is designed and implemented as a set of interactions among services. An application evolves

through the addition of new services. The resulting service-oriented architecture defines the services of which the system is composed, describes the interactions that occur among the services to realize desired behavior, and maps the services into one or more implementations in specific technologies.

We present a study of how to aggregate services provided by a Jabber framework to create an ad-hoc collaborative authoring tool. This paper is organized as follows. In Section 2 we present related work, while in Section 3 we describe our design. Sections 4 and 5 provide implementation details and experimental results respectively. Finally, Section 6 presents our conclusions.

II. RELATED WORK AND CONTRIBUTION

The present work can be compared with works in many different categories, starting with research in collaborative environments [8]. Many tools, frameworks and middleware exist in literature. For example, [12] explores allowing users to collaborate on a multimedia presentation. It requires however an application-specific TCP-based protocol, and the emphasis is placed on the authoring tool that is subsequently interfaced with communication and concurrency control modules. In [10], authors explore the possibility of listening for and distributing user events, reconstructing the interaction with the application and thus sharing Java applications. Instead of using a Jabber framework, an infrastructure presented in [3] can be substituted. It supports development of scalable and evolvable applications composed of loosely-coupled services that communicate via XML document exchange. Finally, service composition in the context of Web Services (WS) is an active research area [5].

To the knowledge of the authors, no previous attempt has been made to develop a distributed multimedia authoring application using Service Oriented Architecture where some services are provided by the Jabber set of protocols. Although the underlying technologies are known, this specific combination of tools and approaches is novel.

III. DESIGN

At its basis, a collaborative authoring tool needs two services, document consistency and group communication. Consistency broadly means ownership management and requires two functionalities, synchronization (common concept of time)

Services	Description
Multicast	A Jabber server usually supports a chat room, which multiple participants can join. Any message issued to the chat room will be forwarded to all participants in that chat room. A chat room can be viewed as a multicast group abstraction
Presence	Each client needs to establish a connection with a server. This connection information can be made available (depending on security and privacy policies in effect) to other clients
Messaging	A client can send short XML-based messages to any other connected client

TABLE I
SERVICES OFFERED BY A JABBER-BASED FRAMEWORK

and causality (precedence relation). Group communication implies both group notifications and bulk data transfer requiring naming, addressing and routing functionalities.

How good those services are will ultimately influence the efficiency of the business process. Responsiveness, the time necessary for the system to respond to users' actions, and fidelity, the degree to which the end result is similar for all users, form the basis of our evaluation criteria.

A. Jabber-based Services

Jabber [1] is a set of streaming XML protocols that enable participants to exchange structured information in close to real time. Two of those protocols in particular are the extensible messaging and presence protocol (XMPP) defining messaging, presence, and request-response services between any two network endpoints, and instant messaging and presence protocol (XMPP-IM) defining instant messaging and presence functionality. Jabber does not require any specific network architecture, but following examples of many instant messaging applications, we are implementing a simple client/server architecture. We are using Jabberd 1.4 software for Jabber servers, and we build our own Jabber client based on the Smack API [2]. As a networking technology, Jabber offers the following four functionalities. *Naming* describes a user's identity. Names are drawn from an infinite space of globally unique Jabber Identifiers or JID. A valid JID contains a set of ordered elements formed of a fully qualified domain name, locally unique node identifier and resource identifier. *Addressing*, which defines globally unique IP addresses of Jabber servers. *Binding* resolves the fully qualified domain name within JIDs into IP addresses through DNS. Local portion of the JID is resolved locally by the Jabber server. *Routing* defines on which connection a packet should be sent. Jabber servers communicate only with other Jabber servers and with Jabber clients it currently serves.

Services offered by a Jabber-based framework are shown in Table I. Note that we are not using certain functionalities such as chat room playback and potential server-side components as they are defined and controlled by the Jabber server administrator and are thus not standardized.

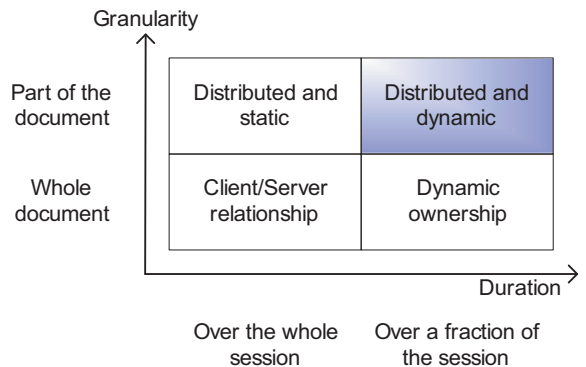


Fig. 1. Design space of document ownership management. The whole document, or part thereof, can be owned by a participant for either the duration or a fraction of a session. Our implementation is distributed and dynamic (highlighted)

B. Collaborative Authoring from Services

We see that the multicast and messaging services will satisfy the need for group notification service. Jabber-based framework does not however provide bulk transfer and ownership management services.

Jabber servers do not handle large amount of data, therefore bulk transfer service needs to be implemented by another framework. A simple web service offering document storage and retrieval (with possibly access control) can be used to disseminate data among clients. We have implemented a web server based solution using HTTP transfer, but without web service interface. Another possibility is a service based on JXTA framework, in order to distribute the cost of large data transfer to participants. We are currently working on this approach.

Without a specialized server-side component (such as a database), a Jabber server has no concept of a document. However, we can leverage the existing group notification service in conjunction with the presence service to implement ownership management distributed over participating users. The design space of such an approach is illustrated in Figure 1. The term *distributed* means that participants share control of all ownership decisions. In later sections we show how the distributed and dynamic solution is implemented.

IV. IMPLEMENTATION DETAILS

The main collaborative authoring functionalities are provided by Jabber Database Access (JDA)¹ and an interactive Jabber Collaboration Interface (JCI) components. Both components are interfaced to Jabber through the Smack API and to the application-specific code. To avoid the inclusion of authoring semantics in JCI, a Document Object Model (DOM) is used to mediate between JCI and the authoring application. The schematic is presented in Figure 2.

Collaborative applications need to strike a balance between two incompatible requirements [11], high responsiveness and

¹JDA allows clients to access a local or remote database. It will not be discussed further due to space limitations.

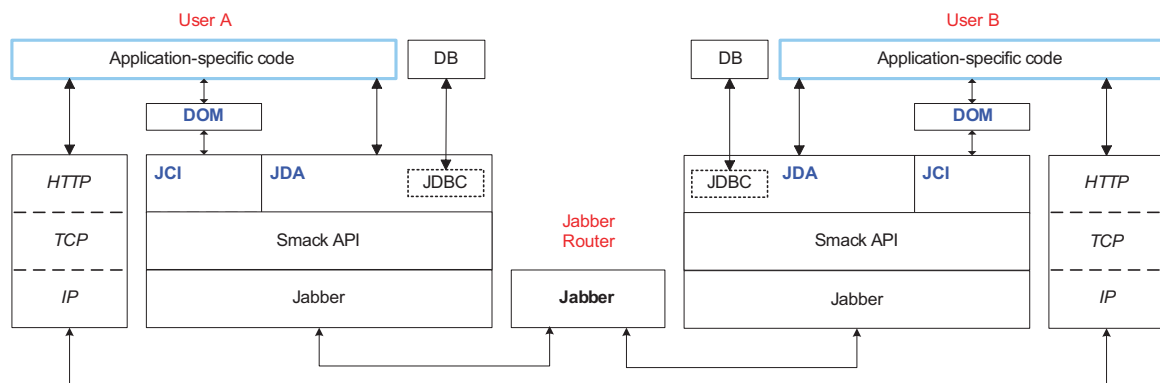


Fig. 2. Schematic of the collaborative multimedia authoring application. Components highlighted in blue are specifically implemented for this application

high concurrency. Perception of quality depends on the timeliness of response of the system to other users' actions, whereas concurrent users' actions cannot result in a globally inconsistent state, even in the presence of conflicts. In our implementation, we focus on the high concurrency requirement - based on group notification, while we assess the high responsiveness requirement from users' application usability perspective. Group notification determines when, what, and how updates generated by one user are propagated and made available to other users [9].

The implemented notification mechanism conforms to the process group model - the distributed system is treated as a collection of users that communicate by sending multicast messages. Ensuring message delivery in the presence of delays and link failures is provided by the Jabber protocol implementation while our implementation addresses the issues of concurrent message ordering and changes to group membership. The JCI component uses the conference (chat) room service provided by the Jabber servers.

A notification policy determines when updates should be propagated, and depends on specific requirements of the application. In our prototype, users' actions trigger an immediate update, if allowed by a locking mechanism. This locking mechanism is a type of floor-control technique [4], which should prevent conflicts from arising. In the rare case a conflict does arise from an attempt to obtain a lock, it is automatically resolved by the JCI component.

The application-specific code contains the implementation of collaborative space metaphors and semantics for user actions such as multimedia object creation, modification and manipulation. This code is also responsible for enforcing all the rules specific to the document being created, including verification if a particular placement of an image will not conflict with another object. The DOM specifies the relationship between multimedia objects and the collaborative space metaphor. The semantics of object placement (e.g. spatial or temporal) need to be defined and understood by all participants prior to collaboration.

Users can add, remove and order multimedia objects. Ordering can be spatial, temporal or both, as defined by the

objects' characteristics and collaboration goals. Objects can be private, or obtained from another participant using the JDA component. Interaction between users takes place in the collaborative space. This is where users become aware of other participants' actions and contributions. This space is application-specific and there should be no restrictions on the metaphor used to represent this space. To keep things simple, our prototype application allows the user to add and order multimedia objects spatially on a 2D finite plane in such a way as to match (in a puzzle-like way) what other participants are doing.

The document object model (DOM) is an XML file specifying the relation of multimedia objects to the collaborative space metaphor. For instance, an image will occupy a certain area at certain position of a specific surface on a 3D object. For temporal objects, another variable would specify the duration of this relationship. Whatever the application, the underlying JCI component can still be used since the DOM describes where conflicts may occur, and as such, is the interface between the application and the JCI component. For testing purposes a DOM that defines sixteen areas to which images are added is used, and as the end result of the collaboration, a collage of pictures is obtained.

A. Collaboration Protocol

In order to maintain a consistent multimedia document in the collaborative authoring context, any modifications made in the collaborative space by a user must be coordinated with the actions of all other users. This coordination results from all participants granting or denying a lock for the part of the document a user wishes to modify. If the modifications are accepted by all the users, a lock is achieved, and a modification takes place. This can be seen by all participants after an unlock message. The collaboration protocol implemented in the JCI component conducts the locking and unlocking of spatial areas as defined in the DOM.

The collaboration protocol uses time-stamps to resolve any conflicting lock requests. For each user, the JCI component maintains a list of locks that are currently in place for all users along with a list of pending requests that this particular

user has made. Requests and replies are carried by group chat and individual messages respectively. There are five types of messages that the protocol defines and accepts. These message types are lock request, accept, deny, lock confirm, and unlock.

1) *Time Synchronization*: The time-stamp on each lock request must follow a standard time that all participants use. This will ensure the time-stamps can be compared in the event of a conflicting request. When two request are made for conflicting areas, we determine which message has the earliest time-stamp. In order for the time-stamp comparison to be relevant, it is necessary that users agree on a standard time. Moreover, it is important that this agreement be not jeopardized by factors such as clock drift or any change in the system time. In order to get all the nodes to agree on time, we use a light time synchronization protocol based on Cristian's algorithm [6].

One client node acts as time server, or leader. The leader sends out a series of values issued from the java call `System.nanoTime()`, whenever a user joins the conference room, or every two minutes; other nodes do not request them. Every user in the room, even the leader itself, then takes whichever time-stamp came through within the shortest delay out of the series, and uses it as a reference from then on when a time value is required. Having each recipient choose the time-stamp that was the quickest to arrive mitigates network jitter.

2) *Leader Election*: In order for the nodes to obtain a reference time, leader election is required. If a user who joins the conference room (the designated Jabber conference room) is the first participant in the room then this user is designated as the leader. If there are other participants in the room, the newly joined user will wait for a leader declaration message. Once the newly joined user has received this message, the latest version of the document object model which represents the multimedia document is retrieved and the collaboration continues.

In the case where the leader leaves the room, all the participants in the room will stop issuing requests, wait for any locks to be released, and then determine a new leader. A new leader is determined by selecting the user who has the highest hash code of their unique user names. Since each user has access to a list of participants in the room then this calculation shall return the same result for all users. Once the new leader has been selected the collaboration can continue once again.

3) *Handling Group Changes*: Group changes occur when a user joins or leaves the conference room and both events are handled similarly. Late join arises when a user joins a conference room that supports an on-going collaboration. This user then needs to obtain the latest DOM, which introduces two issues, obtaining the latest version of the DOM and ensuring that the DOM does not change while the new user is in the process of acquiring it. A file that meets both of those requirements is the authoritative version. Similarly, when a user leaves, its outstanding locks need to be deleted, and an authoritative version posted. Although all users have the same DOM and could therefore provide their copy, it

makes conceptual sense to have the leader responsible for providing the authoritative version. When a user joins an empty conference room, it becomes the leader and retrieves and loads the DOM into memory. From then on, the leader is responsible for posting authoritative versions of the DOM. When a user joins a conference room that is not empty, it awaits a leader declaration message. Before the leader sends this message, it waits for all locks to be unlocked, while all other users stop issuing new requests (replies and time-outs are still processed). When all locks are unlocked, the leader posts the DOM and issues the leader declaration message. The new user then retrieves the authoritative version of DOM from the designated web server.

During the course of collaboration and in the absence of any group changes, processing an unlock message will update the version of the DOM that the user has in memory. Unlock messages hence contain all necessary and sufficient details of the modifications made to the multimedia document.

4) *Deadlocks*: In both the leader election and group changes procedures, clients might have to wait for locks to be unlocked, which presents a potential deadlock problem. If a client shuts down or its connection to the Jabber server is severed (due to a failure or by design), the server detects that the TCP connection to that client has been closed, and assumes the client has left. The server then sends a presence notification to the other participants, which allows clients to discard all pending and active requests owned by the disconnected client. If, however, a client *hangs* (e.g. malicious client) and stops responding while the TCP connection is still alive, requests emitted by other peers would continuously time out. To handle this case, other participants would need to agree which client failed and requires an addition to our protocol.

A second case when a deadlock can occur is when a client connects to the server and receives a message saying it has joined the group, but is not yet in the group when requesting the list of participants. This is an artifact of the Jabber server implementation found out during the development of our prototype and happens very rarely. Although not implemented, this case could be handled by ensuring that the client cannot proceed if it is not present on the conference room roster.

V. EXPERIMENTS

We have simulated seven distant users collaborating over the Internet and the evaluation is based on the following five metrics. First, the average total time to request a lock, obtain, it, upload a relevant multimedia file, and send out the update. When using an actual application built over our component, this metric may correspond to the actual time the user has to "wait after" the application. Second, the average time required to lock an area. This metric is calculated by subtracting the time at which a request was submitted to the time at which it was accepted by every collaborator. This measurement provides a lower bound for the total update time. Third, the delays when a user signs in or off. This "delay" will be considered a measurement of the impact on users, since it corresponds to a period during which they cannot use the

application actively. Fourth, the proportion of non-successful requests. This metric measures the negative impact on users in the form of their changes being rejected by the framework, requiring the user to reconsider the change they're trying to make and possibly redo it. Finally, the variation of the above metrics with a larger group of collaboration. We will discuss the variations in performance while having three to seven users present.

We have implemented a script that simulates the behavior of users collaborating on a multimedia document, and included the following three actions: join a chat room for collaboration and downloading the latest version of the document being worked on, make changes to the document, while updating the local copy when updates are received from other users, in accordance with the protocol and finally, save a backup copy of the document whenever somebody leaves the conference room. Upon launching, our test program automatically logs in and joins a collaboration room hosted on Jabber.org server. After joining, a thread is launched to simulate a user who would drag and drop a new image every seven to ten seconds on the document, onto an area taken at random, among the 16 fixed predefined areas of the testing document. Once the request is accepted and the lock confirmed, the script uploads a random picture to a common HTTP server, through a CGI PHP script. The picture is picked amongst a set of three pictures, with the following sizes and likelihoods of being chosen: Picture1, 35kB in size, 50% of the time, Picture2, 75kB in size, 33% of the time and Picture3, 200kB in size, 17% of the time. Once the picture is uploaded, the area in question is changed to reflect the new object, and an update is sent to peers, who will in turn update their own copy of the document and mark the area as unlocked.

We have created a scenario in which clients use different types of Internet connection, while the Jabber server is the jabber.org server, which, to the best of our knowledge, is located in IOWA, US. The first client was running on a computer located in Gatineau (Qubec), which was hosting our HTTP server. This computer was connected to the Internet via a domestic DSL connection with a downstream capacity of 3.0 Mb/s. This user will be referred to as "User1". Another computer was hosting one user, and was connected to the Internet via the wireless network of the University of Ottawa. The user on this computer will be referred to as "User2". A home network of three computers, running a residential cable connection, hosted up to four users: the first two PCs, connected by wire, hosted User3, User4 and User5; and the third one, a laptop connected over an 802.11b wireless connection, hosted User7. Finally, a fourth computer hosting one user, in Ottawa, was connected to the Internet through a broadband cable connection. The user will be referred to as User6.

A. Results and Discussion

The general observations are that performance degrades gradually with the addition of users, and delays peak when the seventh user joins in. The proportion of requests timing

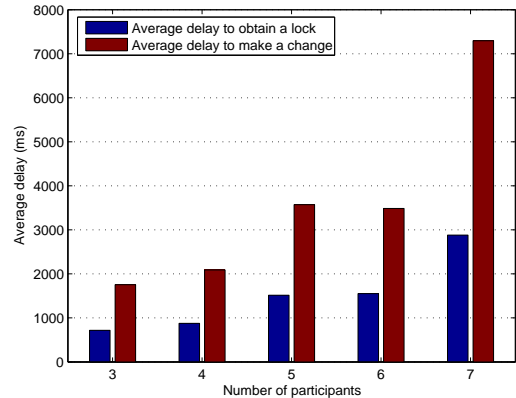


Fig. 3. Average delays to obtain a lock and make a change as a function of number of users.

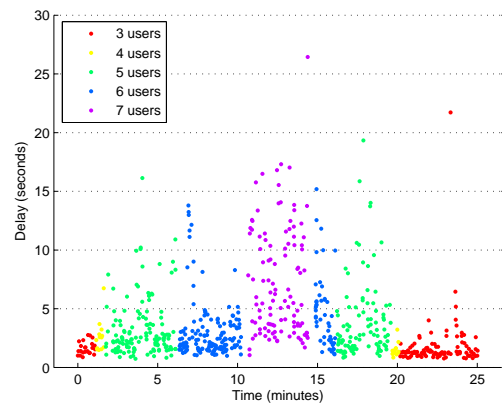


Fig. 4. Time to make a change during the life of the simulation.

out follows this trend as well. Any user that introduces delays (joining the collaboration or downloading) will have an immediate impact on the performance of the other participants.

Figure 3 shows the average of delays to obtain a lock, and total time to make a modification to the document, as a function of the number of users present in the collaboration room. The time to obtain a lock and to perform an update is relatively stable up to six users, with the majority of changes taking less than five seconds as shown in Figure 4. It also shows the per-request total time to make changes to the DOM throughout the life of the simulation, of around twenty-five minutes. It can be observed that the addition of the 6th user actually lowered the average total update time. The system requires up to three seconds on average (with seven participants) to confirm that the area is not locked by somebody else, and a few more to actually perform the change.

The addition of User7 increases both the time to obtain a lock and to perform an update. We have identified three factors that might affect performance in our experiment. First, most public servers, in order to remain reliable and fair to all, use a technique called "choking", and may delay or drop messages

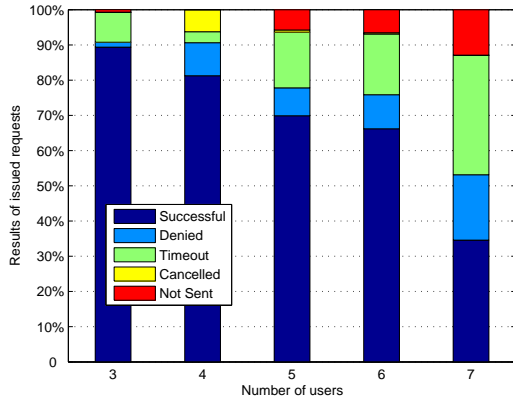


Fig. 5. Results of issued requests as a function of number of users.

from a user who is sending heavily. Second, it is expected that our script attempts to upload, on average, approximately 10 kB/s worth of data, per user. In the case of the computer hosting User3, User4, User5 and User7, running four clients at the same time from the same network implies an average of 40 kB/s in upload. Taking into account that another bandwidth-consuming application was also running on one of the machines (a BitTorrent client) it seems reasonable to believe that the upload bandwidth became saturated. A closer look into the log files confirmed this conjecture, as the upload was observed to take much longer as users were added. This alone directly affects the average total time to perform updates. Finally, with two of our test machines running wireless connections, it is suspected that this might cause some of the observed spikes in delays.

The result of issued request is shown in Figure 5. Requests "not sent" are those that were canceled either because the area was already locked, or because the activities were paused in order to let a user join the collaboration room. Canceled requests are requests that were retracted when another user was logging in. Timeout requests are those for which not all accept messages were received within 5 seconds; those were re-issued. Denied requests are those that were withdrawn because another peer attempted to lock the same area a moment before. Successful requests are those that completed successfully. We observe a constant decline in the proportion of successful requests when more users participate. One of the reason is that our script attempts to make changes to a randomly chosen area. The probability of picking the same area increases with the increasing number of users, in fact if users made requests at the same time, around $1 - \frac{16!}{(16-7)!} / 16^7 \cong 78\%$ of requests would have at least one conflict. In real life, humans might act somewhat less randomly and make lesser amounts of conflicting updates. The increased traffic due to conflicting requests may explain the increased number of time-outs as well.

The impact of users joining in is presented in Figure 6. This delay is introduced by the time necessary to finalize

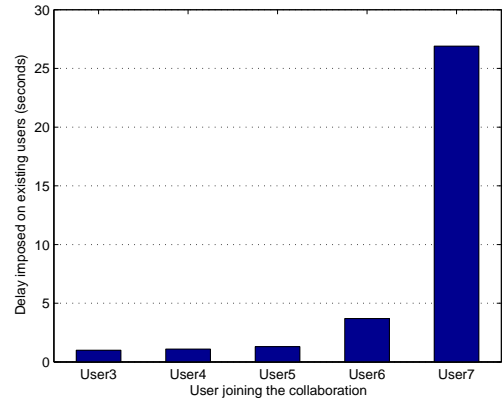


Fig. 6. Delay in seconds experienced by users when a specific user joins the collaboration.

all pending locks, and by the time necessary to upload and download the authoritative version of the DOM. This delay is increasing slowly up to six users. The exception is the addition of User7 which caused other members to stop for 27 seconds. Analysis of log files revealed that it took 23.5 seconds for User7 to download the latest version of the document model, which is not explained.

Finally, at the point where users were joining and leaving the conference room, we compared sample copies of the document model. Inspection of those copies has shown no inconsistencies between users' documents.

VI. CONCLUSIONS

A collaborative authoring application will require certain basic services which can be supplied by various frameworks. We are using the Jabber framework to provide some of those services. Jabber was designed to provide near real-time structured information exchange and thus naturally provides the required group notification service. We are using this service, in conjunction with the presence service to create a collaboration protocol by distributing the responsibility for document consistency to participating users. The implementation of this protocol mandates a strict document consistency, and presents a great scope for improvements by taking the application semantics into consideration. Finally, Jabber does not provide any functionality for bulk data transport, and the corresponding service needs to be provided by another framework. The experiments made with this application highlighted that responsiveness remained reasonable with up to six users when used over the Internet and using a public Jabber server, and no inconsistencies were found in the resulting copies of the document.

Future work includes two main thrusts. The first, is to run the experiments on a single high-bandwidth network and to reduce the probability of conflicting requests in order to obtain a measure of the protocol's performance while in a controlled environment; the second is to compare our implementation to an implementation based on traditional web services standards.

ACKNOWLEDGMENT

The authors acknowledge the financial support of Natural Sciences and Engineering Research Council of Canada and the Government of Ontario (PREA).

REFERENCES

- [1] www.jabber.org.
- [2] www.jivesoftware.org/smack.
- [3] R. Agrawal, R. J. B. Jr., D. Gruhl, and S. Papadimitriou. Vinci: a service-oriented architecture for rapid development of web applications. In *World Wide Web*, pages 355–365, 2001.
- [4] H.-P. Dommel and J. J. Garcia-Luna-Aceves. Efficacy of floor control protocols in distributed multimedia collaboration. *Cluster Computing*, 2(1):17–33, 1999.
- [5] S. Dustdar and W. Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005.
- [6] C. Fetzer and F. Cristian. An optimal internal clock synchronization algorithm. In *Compass '95: 10th Annual Conference on Computer Assurance*, pages 187–196, Gaithersburg, Maryland, 1995. National Institute of Standards and Technology.
- [7] M. P. Papazoglou. Service -oriented computing: Concepts, characteristics and directions. In *WISE '03: Proceedings of the Fourth International Conference on Web Information Systems Engineering*, page 3, Washington, DC, USA, 2003. IEEE Computer Society.
- [8] E. M. Schooler. Conferencing and collaborative computing. *Multimedia Systems*, 4(5):210–225, 1996.
- [9] H. Shen and C. Sun. Flexible notification for collaborative systems. *Proc. ACM Conference on Computer Supported Cooperative Work*, November 2002.
- [10] S. Shirmohammadi, A. E. Saddik, N. D. Georganas, and R. Steinmetz. Jasmine: A java tool for multimedia collaboration on the internet. *Multimedia Tools Appl.*, 19(1):5–19, 2003.
- [11] C. Sun and D. Chen. Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer-Human Interaction*, 9(1):1–41, March 2002.
- [12] M. Sung and D. Lee. A collaborative multimedia authoring system. *Lecture Notes in Computer Science*, Springer-Verlag, 3033:311–318, 2004.