Synchronous Collaboration in CmapTools

Technical Report IHMC CmapTools ??-??

Alberto J. Cañas, Niranjan Suri, Carmen Sánchez, Juan José Gallo, Sofía Brenes
Institute for Human and Machine Cognition
40 South Alcaniz St., Pensacola FL 32502
www.ihmc.us

Introduction

The *CmapTools* software suite is an environment that enables and encourages collaboration and sharing in the construction and manipulation of *Knowledge Models*[1] based on concept maps (*Cmaps*). Collaboration must be provided through useful tools that users can benefit from, and that will allow them to share and expand their knowledge. One of these tools is the synchronous collaboration, which allows concurrent editing of *Cmaps* by multiple users, where they can see the changes others make in real time. It is a complement of the Knowledge Soups and Discussion Threads capabilities, other collaborative tools included. Consider a situation where members of a team need to work on a *Cmap* and they are not in the same physical location. They can collaborate on the construction of their map in real time, avoiding the hassle of having each of them construct a *Cmap* separately and then try to merge the knowledge and information. They can have faster and better results using the synchronous collaboration tool.

The technical aspects of design, and implementation of synchronous collaboration will be further discussed in this report.

Motivation

Connectivity has gained considerable importance in the past few years, resulting in the fact that almost every computer user has access to the Internet, its use has become more widespread and people are now able to connect at faster more reliable speeds more frequently. One of the steps

[1] A *Knowledge Model* is a collection of concept maps and associated resources about a particular domain of knowledge.

that has been taken to take advantage of this resource is to collaborate and share information with people all over the world, shortening distances and producing more efficient, low cost work. Collaborative tools have become popular in all sorts of domains, provided for workspaces in large companies, universities, and countless environments.

Given that one of the primary goals of *Cmaps* is to be able to share knowledge, it's only natural to try to include synchronous collaboration on *CmapTools*. This way, the process of constructing a *Cmap* can be shared. Construction is often the most important part, because of the different ideas that can be exchanged in obtaining the relevant concepts and defining the structure of a specific domain, giving the user a "hands on" experience, getting a better comprehension of the subject.

Consider a tutor trying to teach its students how to make *Cmaps*, so they can learn to organize their knowledge. It is much more useful for the teacher to be able to see the way the student constructed the map, rather than analyzing only the final result. Synchronous collaboration would provide a way to achieve this, allowing the teacher to see every move the student makes, and also, share some input when considered necessary. Students might be able to study together or review material by collectively constructing a *Cmap*. They can interact not only on their *Cmap*, but also in the text chat tool provided alongside.

User Model

Synchronous collaboration is designed to be simple and natural to the user. No special or different procedures must be taken to begin a collaboration session, since the software will be aware of when a collaboration session could take place, by monitoring the lock a user gets on a *Cmap*.

Edit Lock on a Cmap

Cmaps are stored locally on a user's computer, or on the servers that can be located in *Places*, from where they can be accessed. A *Cmap* is a resource on these servers, and as such, whenever a user opens and tries to edit one, he or she will attempt to obtain what is called a *resource lock* on that *Cmap.* This means that a particular *Cmap* is being edited by that user, so nobody else should be able to edit it at the same time.

If a user opens a *Cmap* that has already been locked by another user, whenever this new user tries to edit it*,* the software will know that this resource is currently locked. If the Collaboration Module is present, it will know a synchronous collaboration session can possibly start.

A collaboration session

When the Collaboration Module is alerted that a lock was requested on a resource that was already previously locked, it will start taking action. For the purposes of the example, we will now refer to the user that originally obtained the lock as *lockOwnerUser.* The user that tried to get the lock but failed will now be *requestUser.* Normally, after *requestUser* attempted to edit the *Cmap,* the request for the lock wouldn't have been successful, and the message shown in Figure 1 would have been displayed.

Figure 1

Now, with the Collaboration Module present, *requestUser* will be given an option, shown in Figure 2. Here, a user name can be chosen to be used during collaboration. Now, *requestUser* can decide whether or not to attempt a collaboration session. In case *requestUser* decides not to, the message in Figure 3 will appear.

Figure 2

Figure 3

At this point, a new untitled *Cmap* will be created, with the contents of the original *Cmap*, so that *requestUser* can now edit freely but will have to save to a new location or with a new name. If *requestUser* does decide to collaborate, upon hitting the Yes button, Figure 4 will appear:

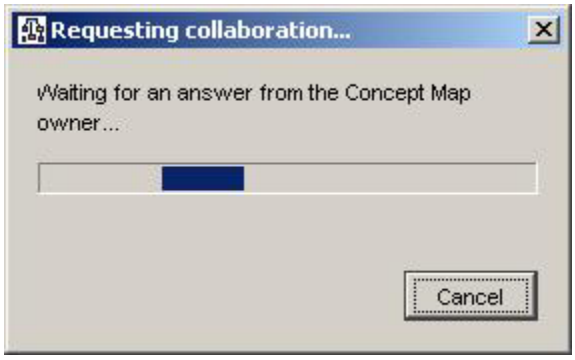

Figure 4

An answer is now needed from *lockOwnerUser,* who will see Figure 5 shown. There, the user name of *requestUser* will be displayed, and *lockOwnerUser* can decide whether or not he/she wishes to allow the session to begin. Pressing the "No" button will end *lockOwnerUser's* participation, and *requestUser* will receive the message in Figure 6.

Figure 5

Figure 6

In case *lockOwnerUser* did accept the collaboration, then *requestUser* needs to get an updated version of the *Cmap*, since *lockOwnerUser* has already been editing it, so the message *requestUser* was seeing on Figure 4 will change to the one on Figure 7, while *lockOwnerUser* sends the updated version of the *Cmap*.

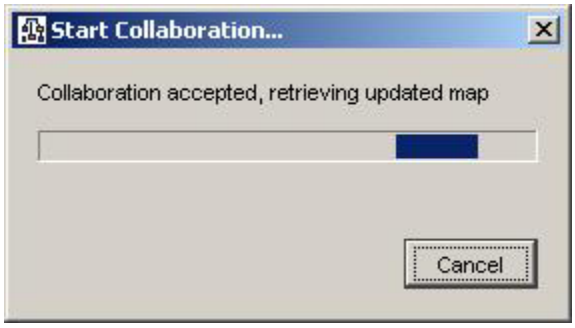

Figure 7

Once the new updated version of the *Cmap* gets to *requestUser*, it's considered that the Collaboration Session has started. A message like the one on Figure 8 will appear on *lockOwnerUser* alerting that a new user has joined, and will appear on every user in the collaboration session every time somebody else joins. Now, *lockOwnerUser* becomes the session master for this Collaboration Session, and any other new users that wish to join this collaboration session will have to go through the same process *requestUser* went through, and the session master will have to accept or deny any other further requests of collaboration, in order to keep access under control.

Figure 8

Every user on a collaboration session will see the chat window in Figure 9, through which they can exchange text messages and see the names of the other users connected to this session. The title of the window will indicate what collaboration session the chat window belongs to, identified by the name of the *Cmap* being collaborated on, since a user can collaborate in any number of sessions at the same time.

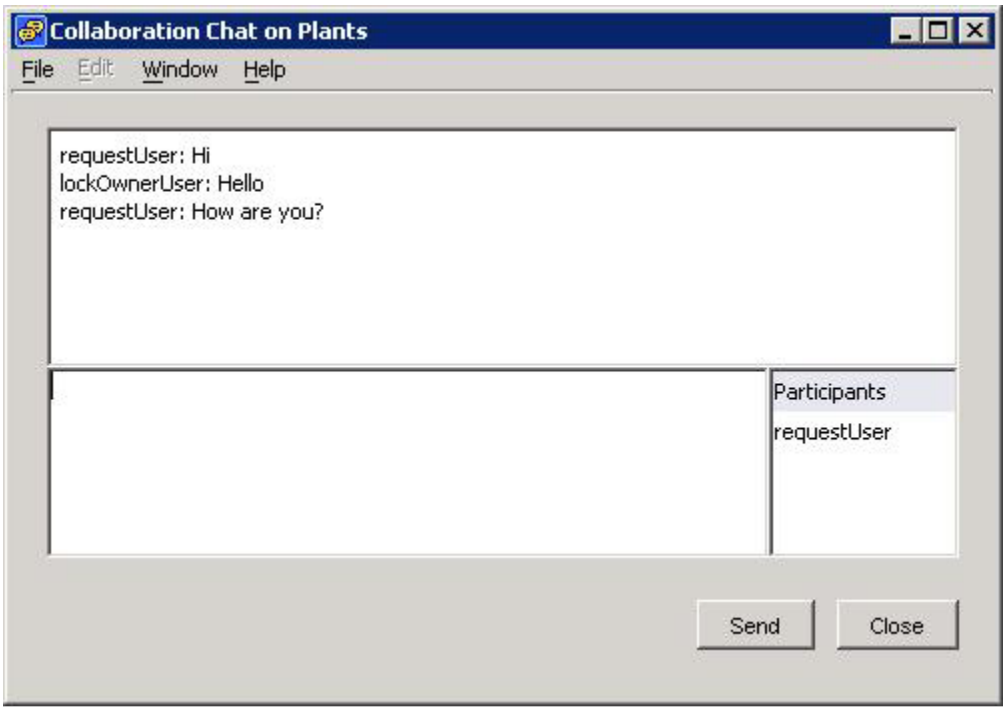

Figure 9

Now they can start editing the map simultaneously, the actions of each user highlighted with a different color, with the user name visible next to the actions taken by each person, so that it's easy to identify the changes that are being made. Any user can save the *Cmap* at any time during a collaboration session. Figure 10 illustrates a collaboration session where three users are modifying a *Cmap.* Collaboration will continue until users start leaving, where the message on Figure 11 will be shown. When the last user leaves, the collaboration session is closed, and the lock on the *Cmap* is released.

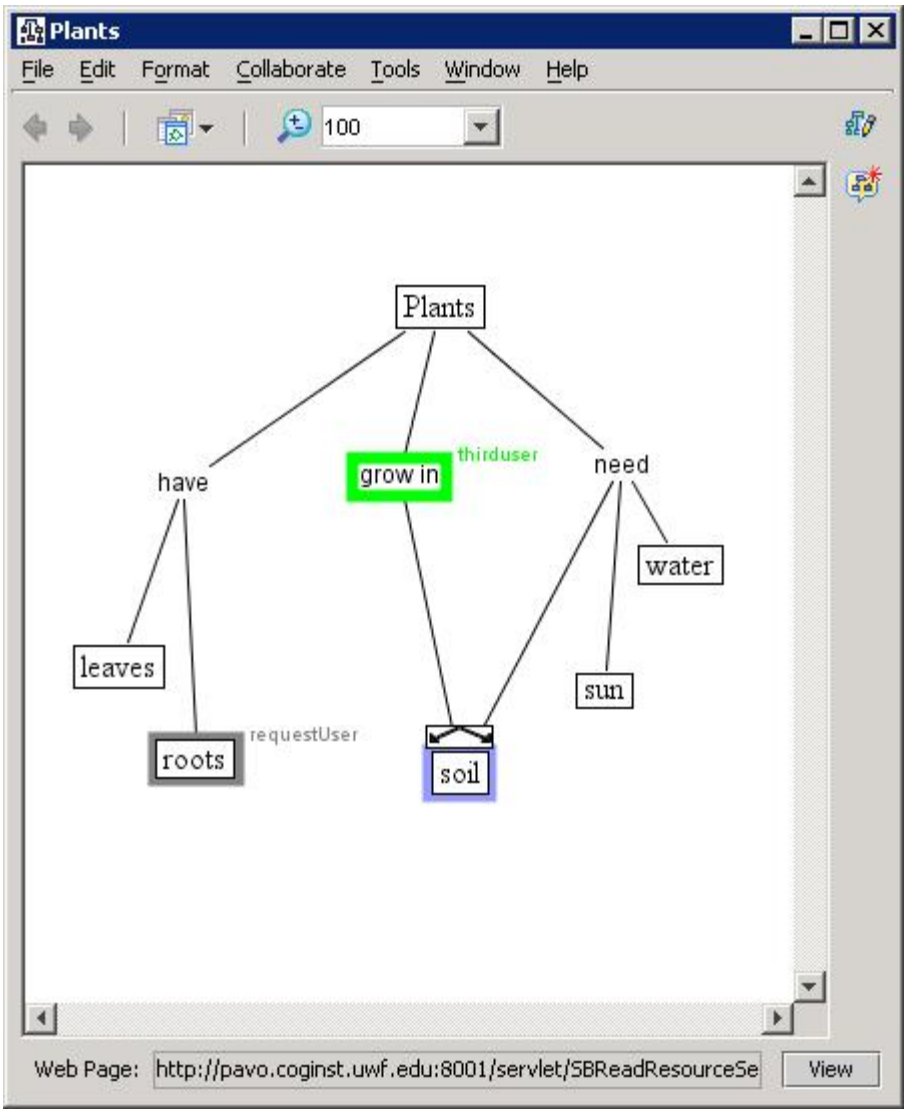

Figure 10

Figure 11

Design

A *CmapServer* is seen in the *Places* and contains resources, folders, and *Cmaps*. They have several basic services and provide an interface called *CmapService* through which new services can be added easily to a *CmapServer.*

Synchronous collaboration is implemented as one of such services. Whenever a *Cmap* is stored on a *CmapServer* that supports the Collaboration Service, it is possible to have a collaboration session on such a map. If the Collaboration Service is not present, the user will simply get a message like the one in Figure 1.

The synchronous collaboration protocol is opportunistic and does not prevent users from modifying the same part of the Cmap concurrently. Even if there is a collision (such as two users grabbing the same concept and moving it to two different directions), the system will ensure consistency by making sure that all the clients contain the same state at then end of the operation. If a client is in the process of modifying a concept, and another user selects that concept, it will be "stolen" from the first user.

Collaboration Client

A *CmapTools* client is able to collaborate if it has the Collaboration Module and the Collaboration Client. These classes will provide communication with the Collaboration Service on any server, and proper handling of the messages exchanged between them.

When a client starts a collaboration session through the procedure described before, a Client Communication Handler is created. This is going to be the communication channel with the Collaboration Service, and it's created using the *CommHelper* and *RequestHandler* classes, which are part of the client/server communication protocols provided by *CmapTools*. Here, messages will be sent and received using a TCP socket connection.

Messages are received and handled according to their type, which include among others, clients leaving or joining sessions, receive and request an updated version of a *Cmap*, text chat messages, and the most important messages, the Action Messages.

The Client Communication Handler can now communicate with both the Collaboration Service and the Collaboration Client, sending to the Collaboration Service the actions generated by the

client, and receiving the messages from the other users so they can be replicated. Regardless of how many collaboration sessions a client is part of, there will be only one Client Communication Handler per client. Figure 12 visually describes the communication between both parts.

Collaboration Service

Given that the Collaboration Service is an extension of a *CmapService,* clients can establish communication with it by way of the *RequestHandler* and the *CommHelper.* Whenever a client issues a "talk to service" command through the *RequestHandler,* the client will be given a *CommHelper* object which it can use to communicate with the service. At the same time, the service will receive that same *CommHelper* object, and it can then store it.

In the particular case of the Collaboration Service, whenever a request is handled, the *CommHelper* will be used to create a Collaboration Client Handler which will be the communication channel between the service and a particular client, analog to the Client Communication Handler on the client side. There will be as many Collaboration Client Handlers on the Collaboration Service as clients collaborating on the *Cmaps* on that server.

The other objects handled on the Collaboration Service are the collaboration sessions themselves. This holds a reference to the *Cmap* that is being collaborated on. It also has the list of Collaboration Client Handlers that are connected to it at a time, so that it knows who its users are. The Collaboration Service controls these sessions, and is in charge of creating and destroying them. Each session has a message queue where all the action messages of every client are placed to be broadcast to the rest of the clients connected to that particular session.

Using this scheme, each Collaboration Client Handler is in constant communication with the client. It receives messages and manages them according to their type. The Collaboration Client Handler holds a reference to the Collaboration Service that owns it, so that it can communicate with the service, and the sessions that it controls. Figure 12 depicts the communication between the client and server side.

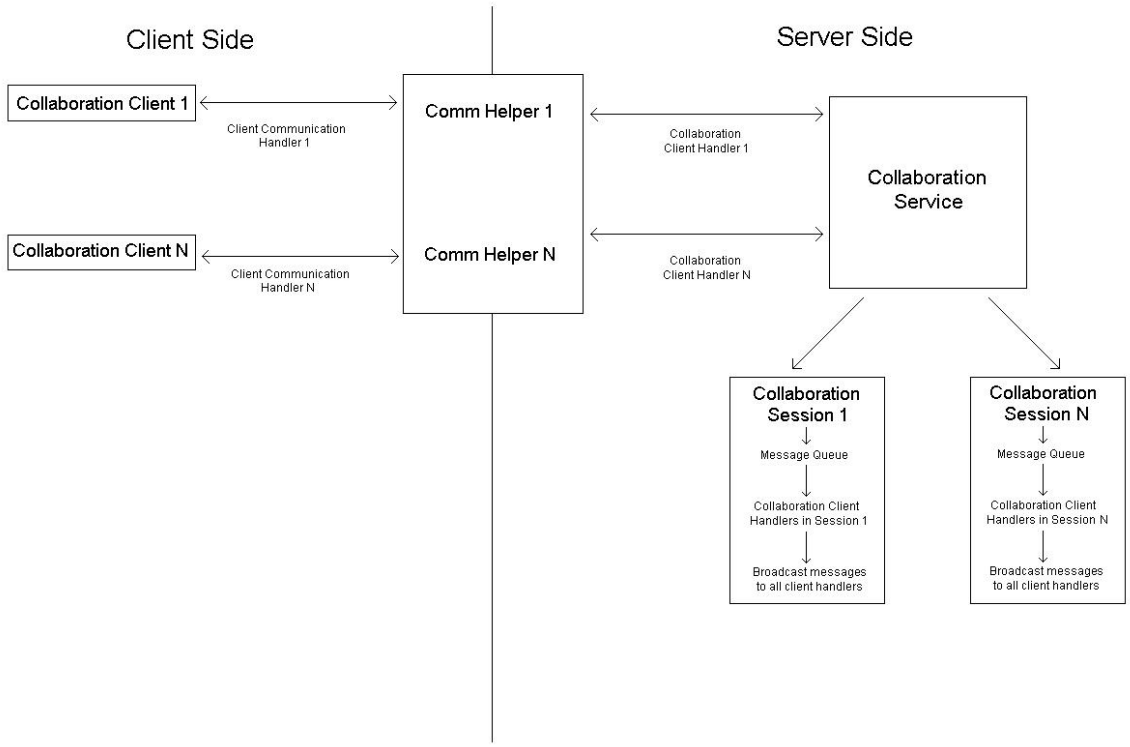

Figure 12

Implementation

Two different approaches have been used to implement the communication between the clients and the service. The first approach used Java Remote Method Invocation (RMI), which was found to be too inefficient. The current approach uses TCP sockets through the *CommHelper*, which results in reduced lag times as well as improved performance over low-bandwidth connections.

The architecture and design of *CmapTools* benefited the implementation of the synchronous collaboration tool, since it takes advantage of the fact that events are posted to produce actions, and listened to in order to handle the results of those actions.

Collaboration Client

The first thing the Collaboration Module will do is make its presence known to the application. It will do this by intercepting the events that are posted when a user is trying to get a lock, and place a flag in it stating that the client is able to collaborate.

After collaboration has started through the protocol described before, using the *RequestHandler* and the *CommHelper,* the *lockOwnerUser* will be asked to transfer the lock of the *Cmap* to the Collaboration Service. This will indicate that a *Cmap* is under collaboration, and will make it easier to keep the lock on the C*map* in case *lockOwnerUser* were to leave the session.

Messages

During a collaboration session, communication is done sending and receiving messages from client to server. These messages are common serializable structures shared by both the service and the client and contain at least the identification of the client who sent it, and the resource identification of the *Cmap* from where the action was generated. They contain different information determined by its type, which in turn determines the action taken by either the client or the server when a certain message is received.

The types of messages are:

- Message: This is the basic type of message and the one from which all others inherit. It contains the identification of the client and of the *Cmap*.

- Action Message: Sent with every action that edits the *Cmap*. These are the most important messages sent, since they represent the edition changes made on the map by various users.

- Chat Message: Send with the text chat messages.

- Client Joined Session Message: Every time a user joins the collaboration session.

- Client Left Session Message: Every time a user leaves the collaboration session.

- Collaboration Request Message: Sent to the session master of an session that has already begun when a new user wants to join.

- Connect Client To Session Message: The initial message sent by a client to the service asking for the service to register him/her on a particular session.

- Estimate Bandwidth Message: Measures the bandwidth the client has available.

- Estimate Latency Message: Measures the latency that affects a client.

- Get Map From Client Message: Sent from the Collaboration Service to the client when it needs to ask that client to send an instance of its *Cmap.*

- Get Session Clients Message: Will retrieve a list of the clients that are currently connected to a session.

- Incremented Bandwidth Message: Used when estimating the client's bandwidth, will have an artificially incremented size in order to test the user's bandwidth capability.

- Map Closed Message: Sent to the Collaboration Service when a client has closed a *Cmap* that was under collaboration.

- Send Map To Client Message: Sent from the client to the Collaboration Service with the client's updated version of the *Cmap*.

- Transfer Lock to Service Message: Sent from the *lockOwnerUser* to the Collaboration Service when a session starts, in order to transfer the resource lock to the service.

The most important message, the Action Message represents the events that *CmapTools* uses to produce the edition changes in their *Cmaps*. It is then very straightforward to obtain the information contained in these events, send it through the Collaboration Service and reconstruct it on the other clients.

Action Messages contain objects called Actions which represent every possible edition event, for example, Entity Added Action, Entity Selected Action, Entity Replaced Action, etc.

Listeners for events are registered on the client side, so that when the user edits something locally, the relevant and necessary information about the event is captured and constructed into an Action, which is sent to the Service. This Action contains the particular name of the event that was caught, so that when it reaches the other clients of the session, the type of Action and the event name are used to determine exactly what kind of event must be reconstructed, and the information is extracted and filled into the new event.

As an example, when the label of a concept is modified, the local listener would create an Entity Modified Action identifiable as an Entity Text Modified event. It will fill out the identification of the concept object that was modified, the new text that was set to that concept, and be sent. The other clients, upon receiving this, will identify the type of Action, read the event name, construct a Modify Entity Text event, fill out the concept identification and the new text, and post it, and the *CmapTools* architecture will take care of changing the text. This event that is reconstructed will have the Collaboration Module as its "creating" module, so that when the event listener catches this text event, it will check to see if the "creating" module is other than the Collaboration Module. If the creator was in fact the Collaboration Module, than the event is ignored, so as not to send the event again.

The Collaboration Storable Event Handler class holds both the listener for the local events, and the methods that reconstruct them upon receiving Action Messages. If any new edition event were to be added to *CmapTools*, it would be necessary to create the listener and the method to reconstruct them inside this class, know the information that is to be sent, and that should be enough to enable it for collaboration.

Collaboration Service

Implementation on the Collaboration Service is very straightforward, messages are sent between the Collaboration Client Handler, Collaboration Service and Collaboration Sessions as shown in Figure 12. There is, however, a continuous measurement of the bandwidth and latency of each clients that connected to the service, so that the Collaboration Service knows which client has the highest bandwidth, and which one has low bandwidth. This is measured in a scheduled task that is performed every thirty seconds, at which time a message is "marked" as estimating latency or bandwidth, so when the client receives it, it will check if it needs to send back an empty message that contains a time stamp and the size of the original message, or reply in any way that will let the service measure its bandwidth and latency.

When collaboration starts a client must either receive a *Cmap* or send its version of the *Cmap,* which is considered one of the largest messages sent throughout collaboration. Taking advantage of this, bandwidth and latency are measured for these messages, in order to set this as the starting value for the latency and bandwidth measures.

The latency task will place time stamps on small messages before leaving the service, upon arrival at the client, when leaving the client and then, upon arriving back at the service. Client processing time is defined as the difference between the time the message left the client from the time it arrived at the client. The total time is known as the difference between the time when the message got back to the service and the time it left the service. To get a more accurate latency measure, the client processing time is subtracted from the total time, so as to get the amount of time the message spent traveling on the network. This value is then divided by two to get the "one-way" latency for a client. The latency value obtained by a message is averaged with the previous value, so a more accurate estimate is formed with each iteration.

As for bandwidth measurement, it is estimated by dividing the message size in bytes, by the latency that particular message experienced, in seconds. Just as for the latency, the value obtained by a specific message is averaged with the previous value, to obtain accurate measures. In case the latency of a bandwidth estimation measure is very low, meaning that the message could be sent in very little time, a new artificial message called Increment Bandwidth Message will be sent to the client.

This message will use the size of the message that was sent in little time, and increase it by ten times. It will then try to send it to the client. If the latency for this message is also small, then it means that a yet bigger message can be sent. If on the other hand, the latency is considerably more, than the size of the next incremented message will be dropped but still be higher than the last message that was sent in little time, so that the real bandwidth measurement is successfully approximated. There is a limit on how big this messages might become, which is discussed later on.

When detecting a client with low bandwidth, some of the action messages, specifically the ones that pertain with modifying the location of entities, are filtered, so that only some of them (excluding the last message which is always sent) are sent, minimizing bandwidth utilization and speeding up the process of collaboration over slow connections.

Whenever a new client joins collaboration, this client must have an updated version of the map sent to him/her, so instead of asking the session master to send his map, the session will look for the client that has the highest bandwidth measurement of them all, assuming that this client is more capable of sending the map fast, and without causing delay problems for that client.

Whenever a client leaves the session, if that was the session master, then a new one must be selected among the pool of clients. The session will look for the client that has the highest bandwidth and automatically make him/her the session master from that point on.

Every time a message is placed in the queue on a Collaboration Session, its size is obtained and averaged with the other messages that have already been placed in the queue in the past. This measurement will allow each session to obtain the average size of the messages that are being sent in it. This average is used to set the limit of how big an incremented bandwidth message can become, since there is no point in testing a client for more bandwidth than what is actually used in a session.