Implementing an Interactive Discussion Forum

Giuseppe Attardi and Giovanni Zorzetti
Dipartimento di Informatica
corso Italia 40
I-56125 Pisa, Italy
mail: attardi@di.unipi.it
fax: +39 (050) 887-226

Abstract

An Interactive Discussion Forum (IDF) is a tool for supporting interactive discussions involving video and audio communication. A forum is a form of structured discussion, more purposeful than informal communications, such as chats or mailing list; moreover it has an important role in achieving a sense of community and participation in the audience. During a forum the participants debate a pre-defined theme, within bounded time limits. A moderator organises the discussion by establishing the argument structure and ordering the interventions, ensuring that the discussion converges to a decision. We outline the key features of our IDF and illustrate several implementation issues and how we solved them. The project is implemented in pure Java using various media and sharing libraries. Moreover we are using handheld computers to test its suitability in a wireless network.

"Now, dialectics allows us to do three things: mental training, conversations, philosophic science research"
Aristotles, *Topics I.2*

1. Introduction

Since the development of dialectics in ancient Greece, discussion and argumentation have been among the main methods of congnition. Greek philosophy distinguishes two such methods: the first one is based on personal study (*monologic*) and it is typical of analysis and synthesis research; the second is a dialogic method involving two or more participants and called *dialectics* (from the Greek word *dialèghesthai*: to discuss). Not surprisingly the early universities grew in the spirit of exchange and discussion of knowledge [Le Goff].

There are several applications on the Web that support various communication and cooperation tasks. Although the potential of the Web for supporting deep learning and critical thinking have been investigated [Newman 96], its use as a "dialectic" tool is still almost unexplored. We have developed an Interactive Discussion Forum (IDF), a tool that provides communication facilities for performing fruitful and purposeful debates over a well-defined issue. A moderator organises and manages a forum inviting some experts who prepare statements on their position supported with background material. The forum involves an audience whose members can take part in the discussion submitting requests for intervention to the moderator.

Most of the issues we encountered are covered in the area of Computer Supported Cooperative Work (CSCW), therefore we exploited ideas from this field. and we adopt the terminology from the CSCW literature: for instance we refer to an implementation of a CSCW system as a groupware application.

For implementing the IDF we chose a semi-replicated architecture in order to distribute the tasks among the components of the forum. Nevertheless, all the communications goes through a central server, which sometimes acts simply as a "reflector" broadcasting messages or streams to all the participants but is also capable of storing a full trace of the discussion.

One particular feature of the IDF has often influenced our implementation choices. The IDF can be used both synchronously and asynchronously. Synchronous CSCW systems require that the participants be present at the same time [Rodden 92]. Implementing synchronous groupware requires particular care in managing shared resources in order to ensure a quick response time to the participants' actions. In the IDF there is only one speaker at a time controlling most of the shared resources; moreover he is supposed to speak for a relatively long period. The approach we adopted exploits these features so that the IDF can be used even in low bandwidth networks like wireless ones.

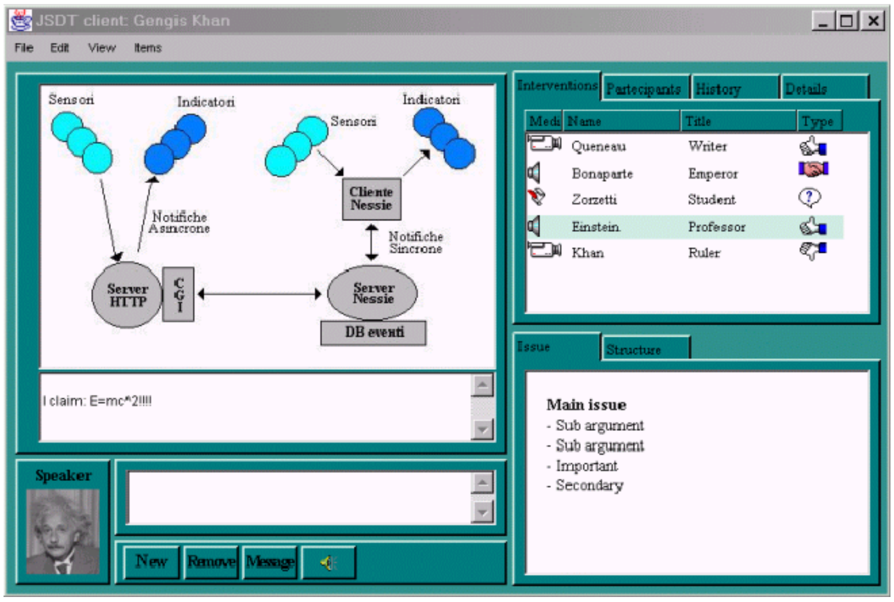
The IDF has been designed with an object-oriented architecture that is highly scalable and that provides in particular a flexible management of message transmission and synchronisation. The various components of the IDF (IDF server, moderator and participant interfaces) are built in Java, enabling participation in a forum by means of any Java enabled device with a network connection, including small handheld devices.
The IDF adopts a flexible approach to message transmission in order to ensure its applicability also in this last case. Where possible and convenient in fact we allow an invisible management of transmission through a byte protocol instead of using the heavier packets produced by the Java object serialisation protocol.

2. Graphical interface

Figure 1 illustrates the graphical interface provided to the participants in the IDF. The top right panel displays information about currently requested interventions, participants and past interventions. Providing detailed information on each participant helps creating a sense of community in the group and improving the cooperation level [Gajewska 95, Greenberg 97].
When requesting an intervention a participant must specify which media he is going to use and he must classify the type of intervention within a fixed number of categories:

- issue
- claim
- supporting argument
- counter argument
- question
- answer
- approval
- disapproval.

The use of dialectic categories is not new in Decision Support Systems [Gordon 96] and in some forum oriented CSCW systems [Eisenstadt 96]. Grouping interventions into categories helps the moderator in selecting the order of interventions and consequently orienting the forum in the most useful way. The structure and articulation of the discussions are visualised in two panels located in the bottom right corner.



I claim: E=mc*2!!!!

Medi	Name	Title	Type
	Queneau	Writer	
	Bonaparte	Emperor	
	Zorzetti	Student	
	Einstein	Professor	
	Khan	Ruler	

Main issue
- Sub argument
- Sub argument
- Important
- Secondary

Figure 1: IDF user interface.

At any time a participant may send and receive messages from the moderator; this dialogue is then memorised and used by the moderator to provide a better management of the forum.
When allowed by the moderator, a speaker can proceed in his intervention using an audio-video connection, two panels for displaying text and images and a file transfer service. An appropriate panel on the left allows visualisation of slides uploaded as images in GIF or JPEG format. The speaker can use a remote pointer or a remote painter (the commonly used terms are *telepointer* and *telepainter*) over the slide to highlight his explanation.

In designing the graphical interface we took into account results from live usage tests for real time applications. In particular we minimised the number of windows used in the interface [Somers 97], keeping all presentation components in one panel. Moreover we tried to give the user a flexible management of these components [Appelt 98].

3. System architecture

The choice of the architecture for a CSCW system is the aspect of its design having most impact on the final performance [Urnes 99a]. There are two perspectives in the forum architecture: the *user perspective* and the *implementation perspective*.
From the user perspective, the only distinction we make is between the role of the moderator and those of the participants: participants are provided with an interface which allows them to direct requests to the moderator and to communicate with other participants, either collectively or privately; the moderator is himself a participant but has an extended set of facilities for managing the discussion.
From the implementation perspective, there are several ways to support the communication infrastructure for the forum. We distinguish between *participants* (including the moderator) when talking about the users of the forum, and *clients* and *servers* when talking about the programs that support the activities of participants. Communication among clients may follow a more restrictive pattern than apparent at the participant level.

In a centralised implementation architecture, a single application program, running on a central server machine, is responsible for most of the tasks and controls all exchanges with the clients. This approach simplifies synchronising clients and maintaining consistent status information (all the data being in the same machine). Besides it enforces an implicit serialisation of events. Examples of centralised system are Rendezvous [Hill 94] and BSCW [Bentley 97]. Replicated implementation architectures instead employ several copies of the same program running on each machine of the system. This increases system complexity, arising from concurrency control management and synchronisation of the clients. On the other hand, the lack of a central server may avoid bottleneck problems, improving particularly feedback and feedthrough response times. An example of replicated architecture is the GroupKit system [Roseman 96].

The IDF has been designed as a semi-replicated or hybrid architecture. The management of shared data is distributed among the *server*, the *participant clients* and the *moderator client*. The server manages data concerning participants and interventions, and controls clients' synchronisation. The current speaker controls the graphical presentation panel and the audio channel. The moderator is responsible for the management of the structure and issue panels.

Many groupware applications with a semi-replicated architecture maintain a distributed approach in communication management. For instance, in the Mushroom system [Kindberg 96] the client must report an event to both the servers and the participants (Figure 2). Since only messages to the server are sent through atomic delivery, this can easily lead to inconsistencies in clients' states. For this reason servers maintain persistent copies of the data used by clients to correct their state.

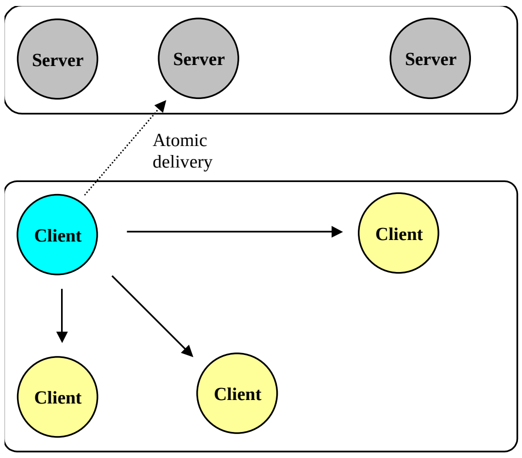


Figure 2: architecture of the Mushroom system.

Groupware applications that support a small number of participants and a high grade of synchronisation commonly use a different approach: one machine is used to reflect the event received from a client to all the participants. The IDF uses a similar model of centralised communication (Figure 3): each client communicates only with the server, which is responsible for multicasting the message to the other clients. This has the additional benefit that future versions of the IDF may exploit protocols supporting multicast at the network level to achieve efficient communication among the participants in the forum.

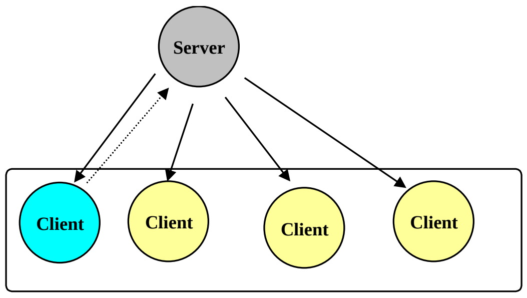


Figure 3: IDF architecture.

A shared bi-directional channel, called “commands”, constantly connects the server to each client. In the IDF there is no direct communication among the clients, not even if a message is sent from one client to another. For example, in order to communicate with the moderator a client must send a message to the server that redirects it to the moderator. Most messages received by the server through this channel are not simply reflected, but cause the transmission of several new messages. For instance submitting a request of intervention to the server produces two messages: the first, with all the information about the intervention, is sent to the moderator; the second, without comment notes, is broadcast to all the participants.

The use of the server for dispatching communications may raise concerns of network delay, since it involves two steps for each message and may affect the suitability of the tool for real time communication. However, exploiting specific multicast support at the network level may offset this drawback and improve significantly overall network performance [Sola 98]. Moreover, directing communications through a central node simplifies synchronisation of clients.

The IDF supports three more pairs of channels used for the presentation of an intervention. One pair is used to receive and send audio data, another one deals with image management and the last two channels are used for additional real time information (such as the mouse pointer).
While the participants directly control the opening and closing of their audio channel, the management of the image channels is done by the system. The server is constantly listening on the first channel, used for the transmission from the client to the server. When it receives some data, the server sends a message on the commands channel inviting all the participants to open the second channel. As soon as all the participants are ready, the server transmits the data and waits for clients to disconnect. Finally the server sends a message of completed request to the orator.
To achieve synchronisation of clients we use a *token* object. Any client may grab or release the token, while the server can check who are the current token holders. The sequence of the actions performed is illustrated in Table 2.
In order to limit the delays, the token control messages are subject to a timeout. A copy of the slides used by the moderator is kept on the server and can be later used to update clients who had problems in the connection.

The last two channels transmit small amounts of data that must be processed in real time: for instance the movement of the pointer and the key pressure during a chat. In order to reduce network delay, these channels are kept open during each presentation; besides it may be necessary to use UDP channels and deactivate buffering at the operating system layer. Unfortunately this approach may lead to a waste of network bandwidth, due to the overhead of headers in IP datagrams. A better solution is to use a client-side buffer to guarantee a fixed delay.

Client	Server
	Data received form the orator
	Invite all participants to open the receiving channel
Open the channel to receive the image	
Grab the token	
	Check that all clients have grabbed the token
	Send image to client
Receive the image and close the channel	
Release the token	
	Check that all clients have released the token
	Advise the orator of completion of request

Table 1: Steps performed for sending images. Steps involving transmission are in bold.

In any case, the implementation of the IDF requires particular care in the synchronisation of the different parts that compose a presentation.

The overall system architecture is organised as a distributed system in which the management of resources and shared information is split among the different components of the forum. The server collects all the communications, dispatches via multicast messages and streams and performs bookkeeping in order to ensure that the participants are kept synchronised.

4. Shared resources and cache memory

In the IDF only one participant can be speaking at any one time: he controls the image and text panel and is the only one who can send data through the audio-video channel. Other participants can only affect the panels for visualizing the list of participants and of interventions, by sending requests to the moderator or simply joining/leaving the forum. The server handles each request in a different thread and mutual exclusion must be enforced by means of monitors, which encapsulate each shared resource. Although this is a convenient approach, it must be used carefully since it can easily produce deadlocks.

Nevertheless, the danger of information inconsistency is not completely avoided even when monitoring a single copy of the data. A client connection can take a relatively long time and a bad management of the connection phases can cause serious consistency problems.

The IDF creates channels between the server and clients based on IP protocols; nevertheless a Web interface will be used as an access point to the forum [Dix 96]. Each participant connects first to a page on a Web server, which supplies the Java applet that implements the participant client. Through this applet the user can register, browse through current forums and repositories of previous forums. When the participant joins a forum, the forum server brings the client up-to-date on the current status of the forum (participants and booked interventions) and starts streaming any current presentation. Joining an active forum is a complex operation, split into three phases. In the first phase the server creates a client object to represent the new participant and notifies his presence to the other participants. Then the client completes the channels' connection and reports it to the server. Finally the server sends the list of participants and of interventions to the client. Table 1 illustrates the steps performed by a client and a server after an accepted request for joining a forum.

Client	Server
Requests to join a forum	
	Checks login and password
	Creates the *Client* object, representing the participant
	Sends the *Client* object to other participants
	Send the *Client* object to the new client
Completes the joining	
Sends notification of joining	
	Sends to new client the list of participants
	Sends to new client the list of interventions
	If necessary sends the list of images

Table 2: Steps performed by a client and a server after an accepted join request. Steps involving transmission are in bold.

During a presentation, the audience cannot modify the slide used by the orator. This may seem a restriction, but it is coherent with our choice of organising discussions as separate individual interventions. Moreover, since only the speaker has control of the visualization panels, we can improve performance by caching data at each client side. In general groupware applications caching may instead lead to inconsistencies [Urness 99a]. All the slides of a presentation can be sent in parallel with the presentation, creating a cache at each other participant side. Displaying a slide requires just retrieving it from the cache through its assigned ID.
In CSCW systems such optimisation is particularly important: heterogeneity in the clients and the need to maintain some degree of parallelism among the participants, can produce situations where the client with a lowest bandwidth sets the pace for the whole system.

5. Software architecture

We expect that a groupware application will have to evolve through user feedback, therefore we adopted a flexible and extensible approach in our implementation, designing proper abstractions which exploit object-oriented features like polymorphism and inheritance.

In the IDF message transmission is performed through four abstractions, corresponding each to a Java class:

1. a communication manager (MainSender);
2. a message (Message);
3. a channel (GenericChannel);
4. a channel manager (GenericPostman).

Except for audio streaming, all messages transmitted in the system are objects of class Message. This class defines the minimal information needed to send a message (Table 3) and provides basic transmission methods that all its subclasses will inherit.
For sending data we need a suitable subclass of Message, which includes the contents to be transmitted and the method for its management.

Similarly all channels used in the IDF must be subclasses of class GenericChannel, which defines a common interface for all channels. To each subclass of GenericChannel will correspond one class that extends GenericPostman. This class must provide method sendMessage(), which performs actual transmission on the channel.
Finally, class MainSender hides all the above details: each transmission is performed using an instance of this class through one method invocation:

```
instanceOfMainSender.sendMessage(Message)
```

The type of channel and postman used are invisible and their implementation can be modified without affecting any remaining code. Class MainSender uses the ability of Java to create classes at runtime.

Attribute	Description
String descriptor	identifies the type of the message
String postmanType	type of postman
GenericChannel channelToUse	name of the channel
boolean serialized	true if contents of message is serialized
String destinationClient	name of the destination client
boolean toOthers	if true sends the message to everybody except himself
boolean synchronize	if true sends a request to open the channel before transmitting the data
Client senderClient	*Client* object sending the data
String sender	name of the sender

Table 3: structure of Message objects.

To illustrate the steps performed in a message transmission let us consider an example using the subclass defined for a JSDT (Java Shared Data Toolkit) channel:

1) a transmission request is issued by invoking method sendMessage() on an instance of class MainSender. This method receives as parameter the Message which contains the name of the class to be used for actual transmission: in this case JSDTPostman;
2) an instance of JSDTPostman is created and its method sendMessage() is invoked;
3) the method determines from it Message parameter the name of the sender, of the receivers, the JSDT channel to use and other necessary information for the transmission;

4) the contents of the message is encoded as specified in the message and sent through the channel;
5) an integer, representing the result of the transmission, is returned back to MainSender.

6. Optimizing object serialisation

Object serialisation is a feature of Java that allows transforming an object into a sequence of bytes, which can be used to rebuild a copy of the original object. A Java class can be made serializable just by specifying that it implements the interface Serializable, which consists of the methods writeObject() and readObject(). The advantage of object serialization is obvious: if class Message is serializable, it can be sent just by serializing the object itself on a network stream.

The default implementation of the Serializable interface involves storing detailed information, including the types of the object and its attributes. However most of the messages used in the IDF can be encoded just as sequence of bytes. Therefore we override the methods writeObject() and readObject() for such classes of message contents, so that they produce and interpret such sequences. In this way we reduce the overhead of serialization by a factor of up to 4.5. Certain fields in the Message class are used to specify the type of serialization used for its contents. It is a Postman's task to identify such type and, in case, to check its suitability.

7. Video and audio support

Video and audio support are nowadays considered essential requirements in a synchronous CSCW system. Prosody, intonation, pauses or speed in the speech allow an orator to add expressiveness and effectiveness to his intervention. Video streaming, instead, is often used to enrich the awareness of the group rather than to provide more information [Ovidiu 96].

IDF audio channels have been implemented using an early release of Sun JavaSound library. Many features are not yet available in this release and voice support is still incomplete, most notably in the lack of speech compression techniques. For the meantime we implemented an algorithm for silence compression that reduces the size of data to one fourth, enough for voice streaming on Internet. The sound format used is the standard one for digital telephony: a sampling rate of 8 KHz codified with 8 bits each using the aLaw code modulation.

The particular characteristics of the IDF allowed us to avoid typical problems for media streaming on Internet. To achieve an acceptable streaming, real time applications usually introduce large client-end buffers that produce annoying delays. Most real time applications need short response time and, therefore, find this approach unsuitable. Since in the IDF one participant is supposed to use the audio for relatively long periods, we can use large buffers and allow an acceptable streaming audio even in low bandwidth networks.

This solution however raises the issue of synchronisation of all the channels used by the orator during a presentation. Since audio streaming is rendered with a fixed delay, we must postpone the visualisation of the corresponding image or the movement of the remote pointer by the same amount. A simple solution of the problem could be to compute the delay, according to buffer size and network bandwidth, and use such value for delaying the other channels as well. But it is not always possible to compute such delay since it is often caused by the synchronisation introduced by the server (as in the case of image visualisation) and not by the information known by the client.

A more general and flexible approach we are studying is to use a general interface for a Java objects, which allows associating an event with a relative time. Creating time-based objects would provide a way to synchronise several different channels.

We are planning to support video streaming using the RTP facilities provided by the Java Media Framework library developed by Sun Microsystems. In order to use video streams even in low bandwidth networks we want to support different kind of services from full long-term connection to the visualisation of slow-updating frames as already used in other systems [Dourish 91].

8. Conclusions

Internet provides the basic facilities for connecting people who could not meet otherwise and the Web has large unexplored potential for stimulating and improving the collaboration among people.
Most of the groupware applications developed for the Web can be split into two broad categories. Some of them provide opportunities to enrich social life: mailing list, chat programs and web forums, for instance, are used for this purpose. Other groupware applications are geared toward achieving specific results: decision support system, applications for software inspection, meeting room, co-authoring are examples in this class.
The IDF lies in the middle between these two categories. An IDF discussion provides tools suitable for helping a discussion to converge to a final decision, but it can also be used to improve the understanding of an issue or even for teaching.
We have outlined the basic features for a tool of this kind and described some of the main problems we came across during its early implementation. In particular we highlighted how to exploit certain design features to reduce the bandwidth requirements of the system.
We expect to refine our prototype IDF through the comments and experience with actual users.

9. References

[Appelt 98] Appelt Wolfang., Hinrichs Elke and Woetzel Gerd.: Effectiveness and Efficiency: The Need for Tailorable User Interfaces on the Web in Proceedings of the 7th International World Wide Web Conference, Brisbane, 1998.
http://bscw.gmd.de/Papers/WWW7-Brisbane

[Bentley 97] Bentley R., Appelt W., Busbach. U., Hinrichs E., Kerr, D., Sikkel S., Trevor J. and Woetzel G. *Basic Support for Cooperative Work on the World Wide Web* in International Journal of Human-Computer Studies 46(6): Special issue on Innovative Applications of the World Wide Web, p. 827-846, June 1997, Academic Press.
http://bscw.gmd.de/Papers/HICSS-30/HICSS-30.ps

[Dix 96] Dix Alan. *Challenges and Perspectives for Cooperative Work on the Web CSCWI* in Proceedings of the ERICM workshop on CSCW and the Web, Sankt Augustin, Germany, Febraury 7-9, 1996.
http://orgwis.gmd.de/projects/W4G/proceedings/challenges.html

[Dourish 91] Dourish Paul. *A Flexible Architecture for Audio/Video Services in a Media Space* Thecnical Report EPC-1991-134, Euro Parc, Cambridge, 1991.
http://www.rxrc.xerox.com/publis/cam-trs/html/epc-1991-134.html

[Eisenstadt 96] Eisenstadt M., Shum S. Buckingham and A. Freeman A., *KMi Stadium: Web-based Audio/Visual Interaction as Reusable Organisational Expertise*, Workshop on Knowledge Media for Improving Organisational Expertise, 1st International Conference on Practical Aspects of Knowledge Management, Basel, Switzerland, 30-31 October 1996.
http://kmi.open.ac.uk/kmi-abstracts/kmi-tr-31-abstract.html

[Gajewska 95] Gajewska Hania, Mark Manasse, Dave Redell. *Argohalls: Adding Support for Group Awareness to the Argo Telecollaboration System*, ACM Symposium on User Interface Software and Technology, pages 157-158, November 1995.
http://www.research.digital.com/SRC/argo/halls.htm

[Gordon 96] Gordon Thomas F., Nikos Karacapilidis and Hans Voss *Zeno A Mediation System for Spatial Planning*, Proceedings of the ERICM workshop on CSCW and the Web, Sant Augustin, Germany, Febraury 7-9, 1996.
http://orgwis.gmd.de/projects/W4G/proceedings/zeno.html

[Greenberg 99] Greenberg Saul. *Real Time Distributed Collaboration.* (in press), Partha Dasgupta and Joseph E. Urban (Eds.), Encyclopedia of Distributed Computing, Kluwer Academic Publishers, 1999.
http://www.cpsc.ucalgary.ca/grouplab/papers/1998/98-Encyclopedia-Distrib/encyclopedia-realtime-collaboration.html

[Hill 94] Hill R.D., T. Brinck, S.L.Rohall, J.F. Patterson and W. Wilner *The Rendzvous Language and Architecture for Constructing MiltiUser Applications*, ACM TOCHI, 1(2):81-125, June 1994.

[Kindberg 96] Kindberg Tim, George Coulouris, Jean Dollimore and Jyrki Heikkien. *Sharing Objects over the Internet: the Mushroom Approach*, Proceedings of Global Internet 96, IEEE, London, November 1996.

[Le Goff] J. Le Goff. *Les Intellectuels au Moyen Age*, Paris, Ed. du Seuil, 1958.

[Sola 98] M. Sola, M. Ohta, T. Maeno *Scalability of Internet Multicast Protocols* Annual Conference of the Internet Society (INET98), Geneva, July 1998.
http://dxcnaf.cnaf.infn.it/~ferrari/papers/multi/scale/scale.htm

[Newman 95] Newman D.R., Brian Webb and Clive Cochrane. *A content analysis method to mesaure critical thinking in face-to-face and computer supported group learning*, Interpersonal Computing and Technology, April 1995, Volume 3, Number 2, 56-77.
http://wwwparent.qub.ac.uk/mgt/papers/methods/contpap.html

[Rodden 92] Rodden Tom. *A Survey of CSCW System* Cooperative System Engineering Group Thecnical Report CSCW /13/92, Lancaster University, 1992.
http://www.comp.lancs.ac.uk/computing/research/cseg/92_rep.html

[Roesman 96] Roseman Mark. and Saul Greenberg *Building Real Time Groupware with GroupKit, A Groupware Toolkit* ACM Transactions on Computer Human Interaction, 3(1), 66-106.
http://www.cpsc.ucalgary.ca/projects/grouplab/papers/1996/96GroupKit.TOCHI/Tochi.html

[Ovidiu 96] Sandor Ovidiu, Konrad Tollmar. *@Work The Design of a New Comunication Tool*, Proceedings of the ERCIM workshop on CSCW and the Web, Sankt Augustin, Germany, February 7-9, 1996.
http://orgwis.gmd.de/projects/W4G/proceedings/atwork.html

[Somers 97] Somers Pat, Carrie Rudman and Clarke Stevens. *Designing Web Interfaces for Realtime Collaboration,* Proceedings of the third Conference on Human Factors and the Web, June 12, 1997, Denver, Colorado, USA.
http://www.uswest/WebConference/proceedings/somers.html

[Urnes 99a] Urnes Tore, Graham T.C. Nicholas. *Flexible Mapping Synchronous Groupware Architectures to Distribuited Implementations*, Proceedings of Design Specification and Implementation of Interactive System (DSV-IS99), 1999.
http://stl.cs.queensu.ca/~graham/stl/pubs/dsvis99.html

[Urnes 99b] Urnes Tore, Nicholas T.C Graham.. *Flexible Mapping Synchronous Groupware Architectures to Distribuited Implementations*, Proceedings of Design Specification and Implementation of Interactive System (DSV-IS99), 1999.
http://stl.cs.queensu.ca/~graham/stl/pubs/dsvis99.html