An Architecture for Flexible Multimedia Group Management Services

Silvester Namuye, Steven Simpson, David Hutchison
Computing Department, Lancaster University, Lancs LA1 4YR, UK
Email:<namuye, ss, dh> @comp.lancs.ac.uk

Abstract

An architecture for connection management of distributed group applications is presented in this paper. The architecture is intended to encourage a uniform appearance to all group applications to assist in their collective management, whilst being sufficiently flexible to cope with all likely multicast mechanisms upon which such applications may be based. The concept of a *media channel* is introduced as the application-independent instance of a single application, and becomes the basic unit of management in a group user-agent which therefore manages multiple applications.

Keywords: group management, multimedia, media channel, session management.

1 Introduction

There are currently numerous distributed group or multi-user applications, e.g. as described by [Hagsand 91], [Dollimore 92], [Wilde 93], [Pendergast 95], [Gay 95], [Muller 96]. These applications often involve only one or two types of communication medium, such as video, combined audio/video, or game simulation. Combining invocations of several of these applications forms a distributed multimedia group session (or *call* in some literature [Heijenk 94]), where several users (a distributed group) agree to have a meeting (a session) using an array of medium types (multimedia) to support their co-operation. As the number of users and medium types increases, there is clearly a need for some form of session-management facility.

In the current literature [Heijenk 94], it is commonly stated that a multimedia group session involves a set of users orthogonally registered against a set of applications, and all the users can participate in all the applications. In this paper we attempt to show that current models are too restrictive for some of the less obvious configurations, particularly sub-sessions. We present an architecture to model sessions in terms of applications, where the session management is defined to be one such application, in order to avoid potential problems with other models.

2 Motivation

Current distributed multi-user (or group) applications manage their own connections to achieve the multi-user effect [Garcia 93], [Trevor 95]. This often results in difficulty when several applications of different types (e.g. video conferencing, virtual whiteboards) are managed as a single multimedia application. Although several group applications may be combined into one, such as a combined video/ audio/ text/ whiteboard conferencing system, if the combined application cannot have new media added dynamically, then the management problem still remains. These systems therefore present non-integrated applications. Session management needs to be dynamically configurable (with regard to usable media) to allow new media types to be incorporated into existing session-management systems.

A number of session architectures have been defined, e.g. [Heijenk 94]. These rigidly define a session or call as being an abstract object which contains a number of channels of different media types (which we call media channels). Users that connect to a session may participate in each of its channels in various ways: as a producer, as a consumer, as both, or as neither. The precise rules for connection (protocols and restrictions) vary with each architecture. In some, users must participate in certain ways before connections take place. Problems arise when external resources, including other sessions or calls, are involved. For example, by adding a video-on-demand (VoD) application to the call (i.e. by adding a new channel), there is now a new user, who is not one of the original users, who provides a video to be played, and could therefore be seen as a (unique) producer of the new channel. This requires exceptions to be made in the model.

There are also problems with formation of sub-calls and call-call interaction. How do you merge the calls? And how do you extricate them afterwards? What if the calls involve different sets of media-channel types? These issues are not addressed in the current architectures. Furthermore, the existing architectures often define the rules of participation, preventing more complex rules from being used.

The architecture we describe is designed to be open: only the interface for participating is defined - the permissions and constraints are left to the programmers. Also, we do not formally define a session or call (as is the case in [Heijenk 94]), merely the media channels that constitute one (although we do define a session in the form of another media-channel type). We aim to cope with both focused and unfocused application topologies (Figure 1), as well as differing multicast mechanisms.

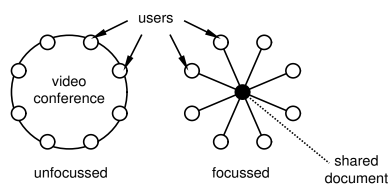

Figure 1: Application topologies

3 Design

Our architecture is built using the basic element, the media channel. This, in turn, is built using streams, which we describe first.

3.1 Streams

A stream is an object of communication between a producer and multiple consumers, distributed across a network. The producer produces a sequence of data (the stream) in a defined format, and this is multicast to each of the consumers, who are expecting the data to be of that format. Each consumer may, independently of the others, ask to receive only a subset of the data by issuing Quality of Service (QoS) requests in the opposite direction to the stream, so that intermediate nodes in the network may filter out the unwanted data. The stream may also stop and start, but always according to rules that the consumers are aware of.

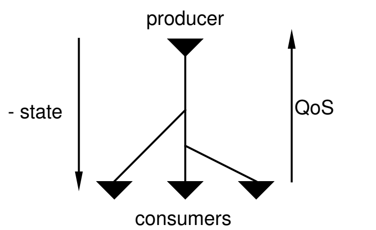

Figure 2. Stream model

These rules and the data format define the *media type* of the stream. An alternative view of a stream is that it is the communication of a state stored by the producer, and there is enough information in the stream such that each consumer can maintain an accurate (if imprecise or incomplete) copy of the state (a mirror). The media type then defines the interpretation of the state, or any of its copies, and how changes in the producer's state are expressed as messages which are communicated to the consumers (we call these messages *delta-state* messages). It must also define the form of the QoS messages. The stream model is illustrated in Figure 2.

3.2 Media channels

We define a media channel to consist of a set of producers (which we call sources), a set of consumers (sinks) and a media type. Each source produces one stream of the channel's media type, while each sink consumes multiple streams - one per source. Therefore, each source is multicasting its stream to each sink. The media channel's state is the aggregate of its sources' states, plus the number of sources. Each sink is therefore maintaining a state which is composed of the (partial) copies of each of the sources, plus the number of sources. All sources registered with the channel must produce state and change-in-state messages that conform to the syntax of the media type of the channel. Similarly, the sinks must be able properly to interpret those messages.

The main requirement of a media channel is that all the sources must send to all the sinks, and conversely, all the sinks must receive from all the sources. We use a *media channel manager* (MCM) to maintain a register of sources and sinks, in a single channel, and this is responsible for ensuring that all (and only) the appropriate streams are established. Sources and sinks connect to the manager to indicate their participation in a channel.

The main aim of the media channel is for each sink to have a mirror of all the sources' portions of the channel's state. For each individual sink, the mirror of any of the sources' state does not have to be perfect: the sink can instruct the source to give it only enough information for a lower-quality mirror, independently of the other sources and sinks - it is able to have the messages filtered.

3.3 Strategy

We define the source-manager and sink-manager interfaces so that all group applications may appear the same, whilst being sufficiently flexible to cope with different transport multicast mechanisms. This allows the further development of call-management applications which handle multiple media types. Also, by allowing the media channel model to run as a stand-alone architecture (i.e. not relying on a call-management application), we are free to allow channels to form part of multiple calls or to be used as independent applications. Writing new distributed group applications now involves defining new media types.

4 Media Channel Configuration and Usage

According to the model defined so far, users may participate in a group application by registering the appropriate source and sink programs with the media channel that represents that applications. A *multimedia* group application is instantiated in terms of multiple media channels, and participation in such an application is achieved by participating in the individual channels. Note that although the organiser of the application, who set up the channels, interprets those channels as forming the complete application, the user is free to interpret them as such, or otherwise. Participation in a multimedia group application does not limit the user to just those channels that form the application, nor is the user required to participate in all of them. We describe two example applications in terms of media channels.

4.1 Applications Scenarios

Video Conferencing (VC)

A video conferencing application can be implemented by a single media channel whose state consists of the images that are currently being captured by the participants' desktop cameras. Each participant runs a source program that produces, say, a H.261 stream generated from the camera, and also runs a sink program that can decode and display multiple H.261 streams. Each participant then sees an array of video images of all participants, including himself.

The media type of the channel will specify which encoding mechanism is used (e.g. MPEG-1, H.261), and what levels of quality a sink can select (resolution, frame rate, etc.). It may also specify that some frames may be 'important'. Sources whose users are currently speaking may mark their frames as important, and users who only wish to see people who are currently speaking may select 'important' as a QoS level.

Video-on-Demand (VoD)

This scenario involves one source (the VoD server) which is churning out a video stream to multiple sinks (the users). This may be represented by a single channel whose state is a video frame, and whose *delta-state* messages form an encoded video stream, e.g. MPEG-1.

Users may specify picture quality levels independently of each other. This does not apply to such qualities as playback speed and which frame to display as this would require the channel to have multiple states. If users wish to share control of the playback speed, a second media channel (the control channel) should be set up to control this quality. Sources of this channel have states that represent their users' choices for this quality, and these are aggregated in a single sink which is able to control the video source. Figure 3 illustrates the VoD scenario.

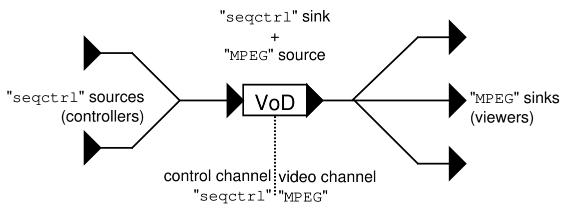

Figure 3: VoD scenario

4.2 User Agent

In order to simplify participation in multiple channels, a user agent may be used to maintain references to channels that the user has expressed an interest in. It is also aware of multiple media types in terms of how to run a source or a sink for each type, and this list of media types is user-configurable. The user may indicate to the agent that he wishes to participate in a channel as a source or sink, and the agent will identify the type of the channel, and run the appropriate program automatically (Figure 4). When the user wishes to terminate his participation, the agent simply destroys the relevant program.

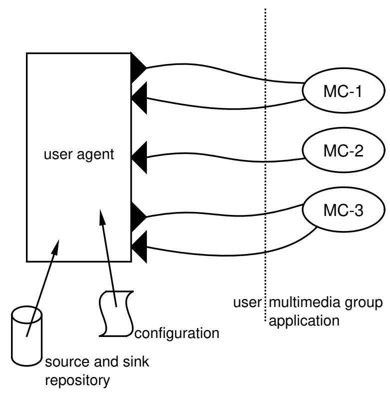

Figure 4: User agent

In summary, the user agent presents a service that allows the user to: retain a reference to a channel that the user may later wish to participate in; to discard such a reference; to participate in a channel as a source; to cancel such participation; to participate in a channel as a sink; to cancel such participation.

4.3 Sessions

In order to participate in a multimedia group application using the user agent, the user's work is considerably reduced, consisting only of entering the references of the relevant channels into the agent, and then indicating how he or she wishes to participate in each of them. This could be made easier by providing a database of the channels that form a complete application, and also indicating how a user should participate in each one (for VC, a source and a sink are recommended; for a VoD video channel, only the sink is useful). This database can itself be seen as a media channel, i.e. the database is the state, whose media type we define to be "session". The multimedia group application is now referred to as a session. The database is initially configured and subsequently altered by connecting a source to the session channel, and a sink program can show the list of component channels to the user.

By combining the session sink and the user agent, participation in a session now only involves entering the address of the session channel, and activating the sink (which is part of the agent). This sink then extends the list of channels held by the agent according to the database held by the session channel, and then begins appropriate participation in each of them automatically. When the user wishes to terminate participation in the session, he or she destroys the sink, which causes all the other channels in the session to be removed along with their sources and sinks.

Note that the session database only offers a suggestion to the agent about how to participate. The user is still free to specify an alternate degree of participation.

As a session is just another channel, it may form part of other sessions, allowing sub-sessions to be defined. This is similar to the method used by [Vin 92] to define nested sessions.

4.4 Security and Participation Rules

As the media channel manager is responsible for connecting the sources and sinks together, it is appropriate to place policies for allowing connection here. If a session is expected to have, say, a minimum number of participants before any conferencing is to take place, the manager may be programmed to defer connection of currently registered sources and sinks until a sufficient number have joined. To implement security within a channel, the manager can be given a list of allowable participants, and may attempt to obtain authentication from them when they join.

Like the session database, the list of permitted participants may be implemented as another channel. Sources of this channel are used to alter the list, while managers who need a copy of the list act as sinks. This enables several channels, such as those that form a single session, to share the same access permissions.

5 Implementation

Implementation is approached by defining base classes of objects to represent sources, sinks, managers and agents, and allowing them to form a distributed system. These base classes form a library from which application programmers can build and configure their own objects.

From this library, we build some example applications to run in the Unix environment: video playback, audio conferencing, and some generic media-channel utilities. A user agent is under development.

The utilities consist of a basic MCM which imposes no restrictions on participation, and the generic sink skeleton which can be configured for a particular application at run-time.

The video application consists of an MPEG player to be combined with the generic sink, and two MPEG sources - one reads an MPEG video from a file, while the other encodes an MPEG stream on-the-fly.

The audio conferencing application consists of an audio source and an audio sink that access the Unix device `/dev/audio`.

All code is written in ANSI C, using structures to represent the distributed objects, which communicate with the manager through TCP/IP or UDP/IP. Communication of the streams is achieved using only UDP, and the sources implement multicasting by making multiple transmissions of the same data.

6 Conclusions

In this paper, we have described an architecture for a flexible multimedia group management services using the media channel model. We have observed that current architectures for multi-user multimedia applications are too restrictive, particularly in the formation of sub-sessions. Our media channel model presents a novel approach that allows us to implement our group session manager as just one more media channel. We have no sense of a privileged user who can control access to the 'call' as is the case in the 'call model' that hitherto has been presented. This is now just a user that can access the participation manager as a source. If there is no such manager, then there is also no such special user. By expressing the access rights as a media channel, we could have any number of special users, i.e. we do not assume that there will always be exactly one. Also, by abandoning the rigid definition of a group session (or call), definitions of subgroups (and sub-sessions) are easier to define because the channels aren't exclusively associated with a single session any more.

7 Further Work

Currently the media channel manager is implemented as a centralised management entity. Future plans are that investigations will be carried out with distributed processing techniques, with the objective of designing distributed media-channel management schemes.

The media channel model may need to be extended to allow for inter-channel synchronisation, such as when an audio-video conference is defined as two separate channels.

Acknowledgements

This work is based within a wider frame of the Group Communications Project (GCommS) funded by the Engineering and Physical Sciences Research Council of Great Britain. The GCommS project aims to formulate an architecture for transport services to support distributed multimedia group applications. Group Management Services Architecture is one of its components.

References

[Dollimore 92] J. Dollimore and G. Coulouris, "The Relevance of Object groups and Multicast in Shared Distributed Object Systems", *In Proc. of Fifth ACM SIGOPS European Workshop*, Le Mont Saint-Michel, France, 1992.

[Garcia 93] F. Garcia, "Continuous Media Transport and Orchestration Services", *Ph.D. Thesis*, Lancaster University, 1993.

[Gay 95] G. Gay and M. Lentini, "Use of Communication Resources in a Networked Collaborative Design Environment", *Cornell University*, Experimental Report, 1995.

[Hagsand 91] O. Hagsand, "A Multi User Draw Editor", *In Proc. of MutiG workshop*, Stockholm, Sweden, 1991.

[Heijenk 94] G. J. Heijenk, X. Hou and I. G. Niemegeers, "Communication Systems Supporting Multimedia Multi-user Applications", *IEEE Network*, (January/February), 1994, pp 34 -44.

[Muller 96] N. J. Muller, "Multimedia Over The Network", BYTE, (March), 1996, pp 73 -78.

[Pendergast 95] M. Pendergast and C. S. Hayne, "Alleviating Convergence Problems in Group Support Systems", *Computer Supported Co-operative Work (CSCW)*, 3 (1), 1995, pp 1-27.

[Trevor 95] J. Trevor, "Infrastructure Support for CSCW", *Ph.D. Thesis*, Lancaster University, 1995.

[Vin 92] H. M. Vin and P. V. Rangan, "System Support for Computer Mediated Multimedia Collaborations", *In Proc. of* 1992, pp. 203 - 209.

[Wilde 93] E. Wilde, "Multi-User Multimedia Editing with the MultimETH System", *In Proc. of 3rd Australian Multi-media Communications, Applications and Technology workshop*, Wollongong, Australia, 1993.