A Time-Sensitive Actor Framework in Java for the Development of Multimedia Systems over the Internet MBone

Giancarlo Fortino[1,2], Libero Nigro[2], Andres Albanese[1]

[1]International Computer Science Institute, Berkeley, CA, USA
[2]Dipartimento di Elettronica, Informatica e Sistemistica,
Università della Calabria, I-87036 Rende (CS) - Italy
Email: {fortino, aa}@icsi.berkeley.edu, l.nigro@unical.it

ABSTRACT

This paper describes an architectural framework for the development of Internet-based multimedia systems such as interactive and collaborative media on-demand applications. The programming in-the-small level centres on Java and a variant of the Actor model especially designed for time-dependent distributed systems. The programming in-the-large level can be tuned to exploit current real-time and control protocols proposed for the Internet MBone. A multimedia application is modelled as a collection of autonomous and (possibly) mobile media actors interacting one to another to achieve a common goal. Multiple stream synchronisation is based on reflective actors (QoSsynchronizers) which filter message transmissions and apply to them application-dependent QoS constraints. Admission control of multiple sessions is delegated to a system Broker. The paper describes the actor framework and discusses its application to the construction of Java Multimedia Studio on-Demand, a multimedia system designed to support playback, recording and editing of multimedia presentations.

Keywords: Java, Light-weight Actors, QoSsynchronizers, Middleware components, MBone, MoD.

1. INTRODUCTION

New advances in computer and network technology make it possible for PCs and workstations to easily support the development of interactive, collaborative distributed multimedia applications. Programming environments, frameworks, toolkits are being implemented and tested to handle, access and deliver digital media over heterogeneous networks.

Current efforts in the Internet research community concern an exploitation of network multicast capability (IP-multicasting) [13], real-time data transport (RTP/RTCP) [26], quality of service (QoS) "guarantees" and resource reservations (RSVP) [28], support for data compression (MPEG, H.261) [28], prioritisation and redundancy techniques (PET) [2].

Challenging problems are involved with large scale network protocols and systems. The main focus is on how to combine building blocks (e.g., media and protocols objects) into the development of complex and scaleable systems.

This work argues that coping with the above technological advances demands, from the software side, the definition of a flexible and scaleable multimedia middleware infrastructure based on object-oriented programming languages and tools. The aim is the achievement of a development paradigm where a system can be assembled in terms of ready-to-use, well-tested and reusable components.

However, the basic components of a design framework must be carefully chosen. For instance, a heterogeneous range of platforms are currently connected to the Internet for use in multimedia applications. They are usually based on general purpose operating systems, such as UNIX or Win95/NT. Such conventional time-sharing operating systems do not provide adequate support for real-time applications involving audio and video manipulation. Without real-time support, timed event processing correctness on the host involved in one or more multimedia sessions cannot be assured.

The concept is to cope with a heterogeneous environment by trying to reduce its complexity through the use of powerful abstractions [3]. This paper claims that a model for open distributed systems like the Actor model [1] has the potential to fulfil the challenging requirements of a middleware infrastructure for interactive multimedia applications. The Actor model has a built-in notion of encapsulation, concurrency and interaction. Moreover, special reflective actors (RTSynchronizers) [25,5,20] can be introduced which capture and verify temporal constraints on message exchanges in groups of actors so as to separate timing from computational aspects.

The adopted actor framework is actually a variant of the Actor model, especially designed for time-sensitive systems. In the small, a set of autonomous and interacting actors are defined, synchronizing and coordinating each other. Actors can be dynamically created. In the large, a system is achieved as a collection of actor clusters, one per processor. Communications can rely on asynchronous unicast and multicast by using both network event passing and mobile actors.

The actor framework provides for both media passive objects (e.g., RTP-Files), media actors (e.g., media streamer, media binder, media session manager and so forth) and QoS synchronizer actors which control message scheduling. Media actors can exploit the services of real-time protocols (e.g., RTP/RTCP) for fulfilling the requirements of continuos media types, and control protocols (e.g., RTSP [27]) for achieving multimedia streaming control in the large.

In order to ensure multi-platform portability, heterogeneous environment interoperability and extensibility, the actor framework is being experimented with the powerful Java programming language which favours clean object design and code mobility.

Section 2 gives an overview to the actor framework. Section 3 describes an actor enabled architecture for multimedia systems. Section 4 shows the design of Java Multimedia Studio on Demand, a multimedia system designed to support browsing, playback, recording, and editing of multimedia presentations. Finally, conclusions are presented which summarise the implementation status and give directions which deserve further work.

2. A TIME-SENSITIVE ACTOR FRAMEWORK

A variant of the Actor model [1] was designed [11] which centres on light-weight actors and a modular approach to synchronization and timing constraints. Actors are modelled as finite state machines. The arrival of an event (i.e., a message) causes a state transition and the execution of an atomic action. At the action termination the actor is ready to receive a next message and so forth. Actors no longer have internal threads for message processing. At most one action can be in progress in an actor at a given time.

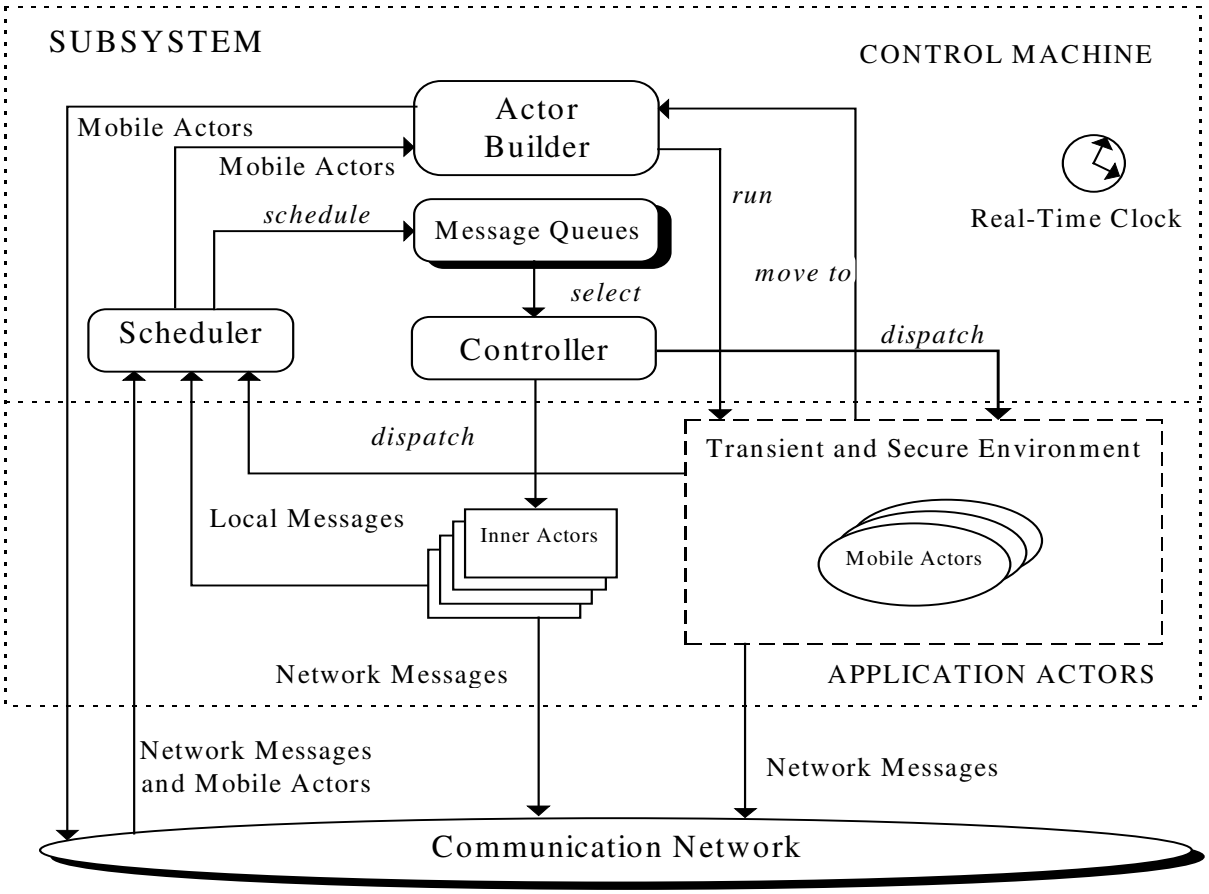

Figure 1: The architecture of an actor based system.

Actors can be grouped into clusters (i.e., subsystems) (see Fig. 1). Normally, a subsystem is allocated to a distinct physical processor. A *control machine* is responsible, on a per cluster basis, of message buffering (*scheduling*) and *dispatching*. The control machine is provided of a time notion (e.g., a real time clock) and can be in charge of scheduling messages according to a timeline. In a simple case, a message can be assigned a timestamp which denotes the required dispatch time. A single *message queue* (plan) can be used for storing all the messages "sent but not yet delivered", ranked by ascending timestamps. The control machine has a method (*controller*) for selecting a message from the message queue according to a chosen control strategy (e.g., minimal timestamp), possibly waiting for the delivery time to be reached, and to dispatch it to the destination actor. At the end of message processing, the controller repeats its loop. The control machine can be customised by programming. For instance, in [11,20] a specialisation of the control machine for hard real-time systems is proposed, where the timestamp consists of a time validity window [t*min*, t*max*] expressing the interval of admissible delivery

times. In this case the message selection process is based on an Earliest Deadline First strategy. In [4] a specialisation of the control machine is shown which is adequate for distributed simulation under Time Warp of complex systems modelled by timed Petri nets.
Within a subsystem, actor concurrency is ensured by action interleaving orchestrated by the controller. True parallelism is possible among actors belonging to distinct clusters.

A distinguishing feature of the actor framework is the modular handling of timing constraints. Application actors are developed according to functional issues only. They are not aware of *when* and *why* they are activated by a message. Timing requirements are responsibility of RTSynchronizers [25], i.e., special actors which capture "just sent messages" (including messages received from the network) and apply to them timing constraints affecting scheduling. A collection of RTSynchronizers constitutes the *scheduler* component of a control machine. Control machines of a distributed system are interconnected by a network and real time protocol. An in-the-large *interaction policy* can be required to have system-wide timing constraints to be fulfilled.

The actor framework can be supported by a few Java base classes in a simple yet efficient way. Actors are implemented as active objects provided of *message handlers*. A message handler implements the state transition diagram of the corresponding actor. The light-weight nature of actors simplifies the realisation of mobile actors [7] (see Fig. 1) which through the Actor Builder based on Java *Object Serialisation* and *Dynamic Class Loader* [14] can easily be streamlined and then reconstructed in the context of the receiving subsystem. Obviously, the mobile actor (sub)environment is required to satisfy security issues constraining the range of things a mobile actor can do. The approach minimises the number of threads in order to ensure a more predictable timing framework [12].

3. MODELLING MULTIMEDIA SYSTEMS

Architecture of a single Multimedia Session

A multimedia session is assigned to an actor system split into two parts specialised to handle the requirements existing at both the server (*transmitter*) and client(s) (*receiver*(s)) sides of the application.

The transmitter side is typically devoted to achieving the multimedia data, e.g., from stored files, and to send them through a network binding to the client for the final presentation. Specific timing and synchronisation constraints exist and should be managed respectively at the server and client side to ensure quality-of-service parameters. To this purpose both server and client subsystems can be equipped with a distinct multimedia control machine with a QoSsynchronizer.
Bindings, i.e., logical communication channels, are introduced for connecting transmitter/receiver subsystems. Bindings can be point-to-point (i.e., unicast) and point-to-multipoint (i.e., multicast). A binding is created by a bind operation originated from media-actors called Binders. A binder governs the on-going flow of data (e.g., continuous media or control messages) sent into the binding. It hides particular transmission mechanisms (e.g., network and transport protocols). It can also monitor the binding QoS so as to provide information such as throughput, jitter, latency and packet loss statistics.

A QoSsynchronizer captures and verifies temporal constraints. As an example, the QoSsynchronizer in a client subsystem can perform fine-grain inter-media synchronisation (e.g., lip sync).

The framework naturally supports event-driven languages for the specification of multimedia presentations [23]. The notion of multimedia presentation is encapsulated in a media-actor called a

Manager (or Supervisor). According to the specification it orchestrates the media objects (time-dependent and time-independent) by interacting with media-actors called Streamers. A Streamer is a periodic actor that accesses to digital media information through media passive object (MediaFile, MediaDevice, MediaNetSource) and sends it to Binders or Presenters. Presenters are media-actors specialised to render media objects.

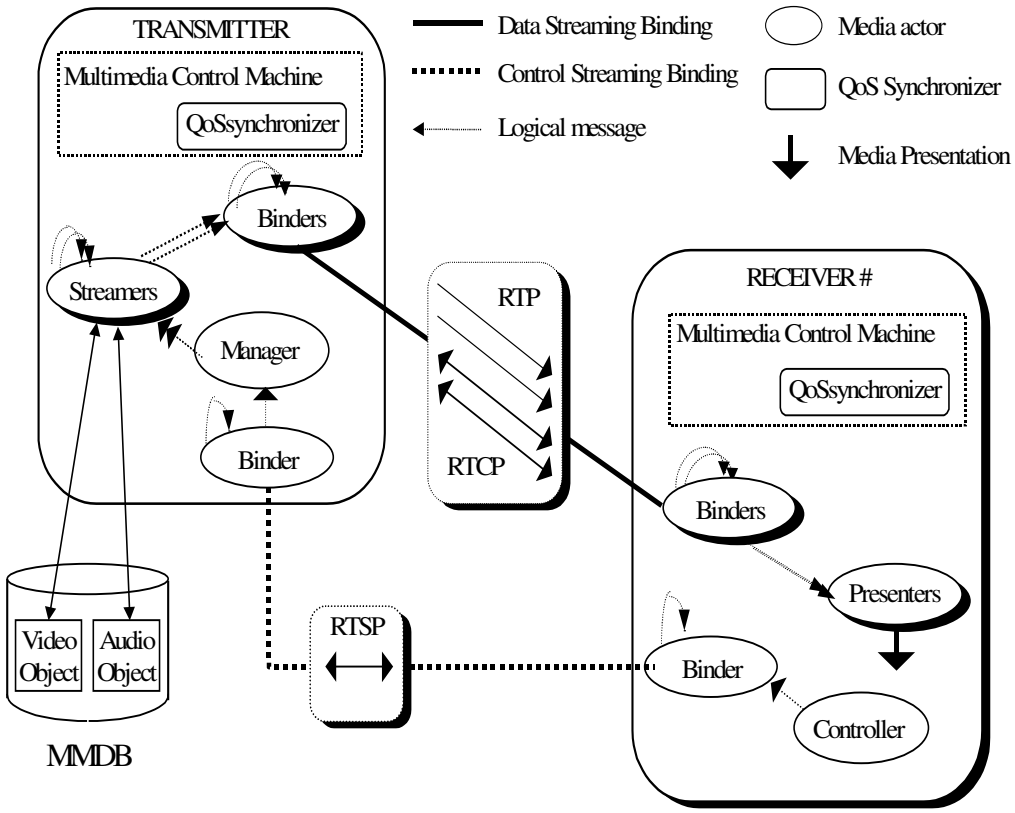

Figure 2 : Playback on-demand of a multimedia session composed of two media stream (video, audio).

Fig. 2 portrays a multimedia system concerned with the remote and interactive playback on-demand of multimedia presentations (e.g., a teleteaching session composed of synchronised video and audio) over the Internet MBone. The Transmitter and Receiver(s) are connected by two bindings: data streaming and control streaming. The former carries the data of the multimedia session according to the RTP/RTCP protocol. The latter makes the interactive control commands flow according to the RTSP protocol. In case the data streaming binding is multicast, receiver subsystems can arbitrarily join the on-going multimedia session requested by its initiator. Normally, only the initiator has the rights to control the session by the streaming control binding. In order to allow a group of receivers to perform control actions on the data binding, a floor control policy [16] should be introduced.

Transmitter subsystem is responsible of the reading process and the enforcement (by using the RTP timestamps) of timing constraints upon the media streams to fulfil the requirements of the multimedia presentation. In addition, it responds to commands (e.g., *play*, *pause*) from the receiver site through the control streaming binding so as to perform an interactive service.

On the remote site, Receiver(s) subsystem(s) control and render the requested multimedia session. The video and audio objects are RTP files previously recorded and archived. The multimedia session is described by the Session Description Protocol (SDP). The presentation description contains information

about the media streams within the presentation, such as the set of encodings, network addresses, inter-stream synchronisation relations and information about the content. At the transmitter site, an actor pair, Streamer and Binder, is instantiated for each media stream. The Streamers read the media files (audio, video) and send them, as messages, to the Binders. At the receiver site, a mirrored situation exists. The Binders poll the bindings and deliver the read messages to the Presenter for rendering purposes.

Multiple Sessions and QoS Broker

Multiple session support is the responsibility of a system-agent called a Broker (or QoSBroker) [18]. It acts as a co-ordinator for all the on-going sessions and performs admission control for new incoming ones [19]. More specifically, the QoSBroker manipulates resources at the end-points, co-ordinating resource management across layer boundaries. Fig. 3 summarises how the various components of a multimedia system can logically be organised according to four different abstraction layers : media, computational, timing, brokering.

Media layer encompasses passive objects that model media information sources such as media files (e.g., RTP, AVI, MOV, AU), media devices (e.g., video capture boards, sound boards) and network streams. Computational layer consists of media-actors such as Binders, Streamers, Managers, Presenters.

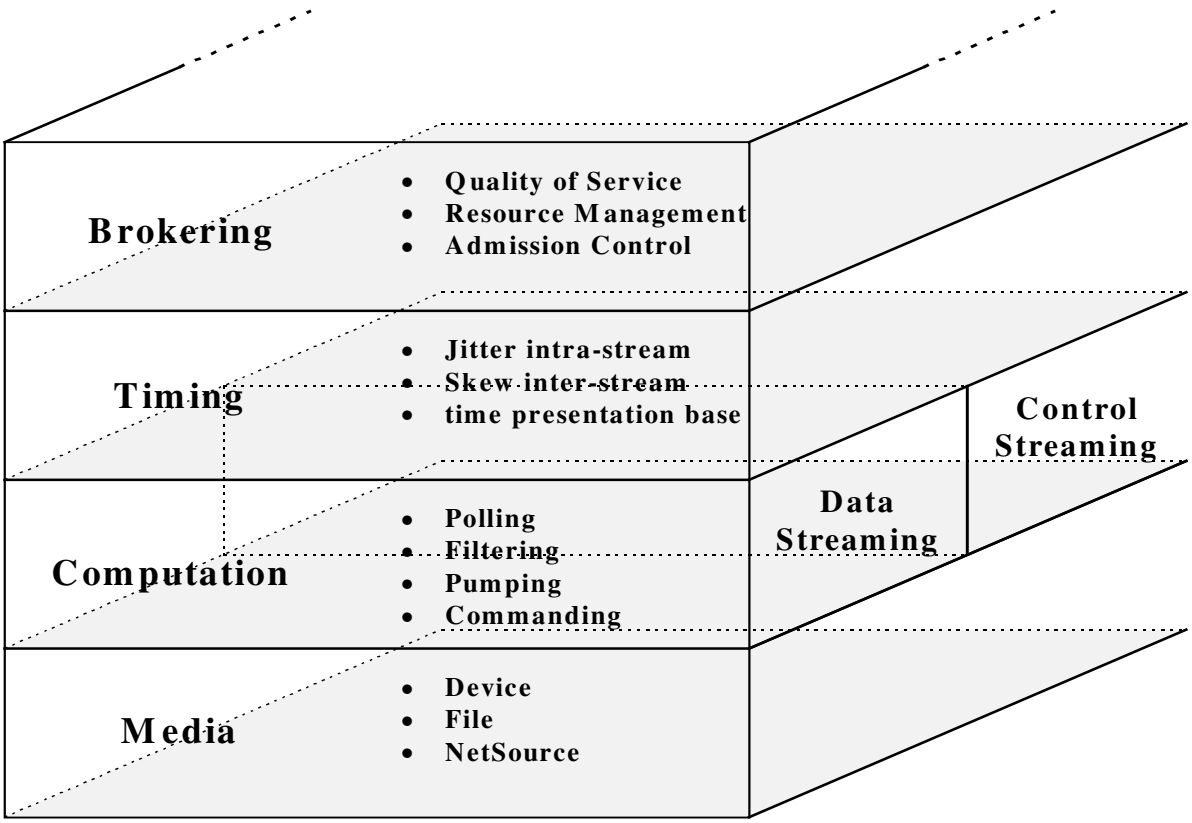

Figure 3: Abstract Layered System Model.

The media-actors Controllers are introduced to handle events generated by the user through a graphical interface. The QoSsynchronizers are located in the timing layer. They manipulate timing parameters such as jitter intra-stream and skew inter-stream. The brokering layer have to do with filtering of user QoS requirements, admission control of multiple sessions and resource management. The system-agent QoSBroker performs the mentioned tasks.

4. JAVA MULTIMEDIA STUDIO ON-DEMAND

Java Multimedia Studio on-Demand (JMSoD) is an experimental distributed system over WWW and MBone for editing, recording and playback of multimedia sessions. It was designed to provide not only interactive presentations (e.g., VoD systems) but also interactive recording and media editing (e.g., media authoring). Interactive presentations mean that the users should have control at least over the following "degree of customisation": the time when the presentation starts, the order in which the various information items are presented, the speed at which they are displayed, the form of presentation. Since the system lives over MBone, it is required to have also the following additional degrees of customisation: the location in the network where the recording or playback takes place, the scope for recordings and playback (e.g., local, regional), the transmission mode for a playback (e.g., unicast, multicast), the users or groups that may access and/or edit a recorded session and the modality of access (e.g., public, private with keyword). The system architecture (fig. 4) consists of Media Servers and Media Clients.

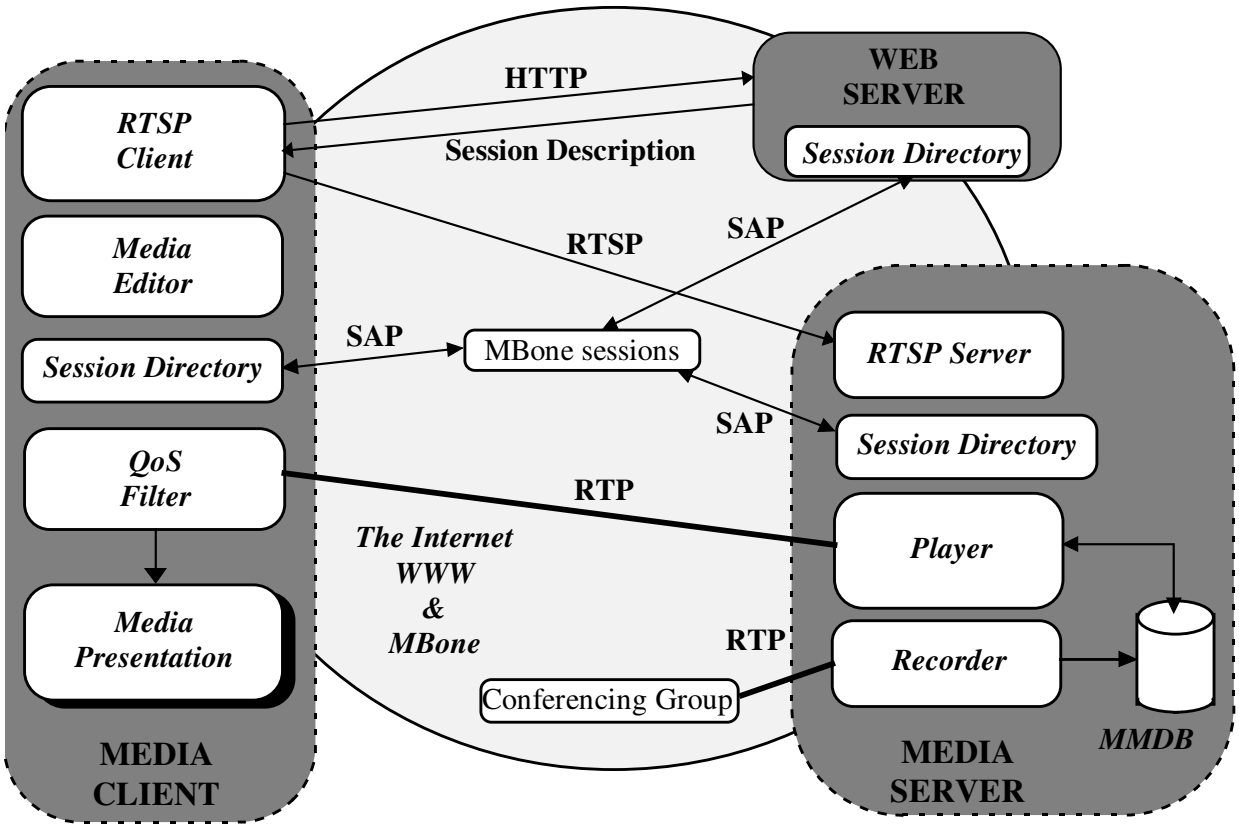

Figure 4 : JMSoD System Architecture.

The media server is the network entity that provides editing, playback, recording and browsing services for multimedia sessions. It is composed of a Recorder, a Player, a Session Directory interface and an RTSP Server. The Recorder records IP-packets on RTP-level. Recording on RTP-level means that it parses the RTP-header of each incoming packet and checks for duplicates and out-of-order packets. It can record data arriving both in unicast and in multicast way. Each recorded data-stream is stored in two files, a data-file and an index-file. The data-file contains the raw RTP packets as in the original session. The index-file provides indexing on the data-file to make random and fast access to the data. A third file

called option-file can be used to store important points or markers in the multimedia session both during a registration and during successive playbacks.

The Player plays packets back according to the Transmitter schema (see figure 2). The Session Directory interface implements an SDR (MBone Session directory tool) using the Session Announcement Protocol (SAP) [9] and the Session Description Protocol (SDP) [8] to listen to and announce multimedia sessions over MBone. The RTSP server implements the server part according to the RTSP protocol specification [27]. The Real Time Streaming Protocol (RTSP) is an application level protocol developed for on-demand delivery control of media streams both alive and pre-recorded. RTSP acts as a network remote control that allows multimedia parts (media servers, media clients) to interact with each other. RTSP is implemented over TCP, since stream control operations need to be delivered reliably.

The media client is the network entity that requests continuous media data from the media server. It is composed of a QoS Filter [7,5], a Session Directory interface, a Media Editor, Media Presentation Tools. The QoS Filter is developed according to the receiver schema (see section 3) and was applied to the lip synchronisation problem [5,12]. The Media Presentation consists of components for rendering video, audio, etc. As an example, they could be VIC for video and VAT for audio. The RTSP client implements the client part according to the RTSP protocol specification. The Media Editor enables the user (or content-provider) to build a multimedia presentation composed of media objects stored in the MMDB by an event-driven specification language [23].

Besides, the system allows for interaction among media clients and among media servers. Tightly-coupled interaction among media clients means that they consistently watch the same multimedia session and operate control on it. Floor control mechanisms [28] (gavel passing, chalk passing) need to be implemented among members of a group in such a way to fulfil consistency. That is, if a member sends a PAUSE command to the Media Server, the others should get the same view. It is the concept of "What You See Is What I See" (WYSIWIS). This is a particular case of the remote control of a shared resource. To hand off control among collaborators, a global co-ordination protocol [17] that manages the device resource or resolves conflicts between different users' control actions is needed.

Media servers can collaborate with each other to perform integrated tasks. An integrated task could be that of playing media streams, each one (e.g., audio or video) from a Media server, and then synchronise them. This can be accomplished in two ways: 1) by choosing a master Media server that co-ordinates all the others (slaves). The master, according to the presentation requirements, asks the slaves for media streams. When the media streams arrive at master site, they are synchronised by using filtering, mixing and translating techniques and sent to the media client; 2) by using a distributed media control between the media servers. After a coarse-grain synchronisation phase, the media streams are sent from the media servers to the media client(s).

Implementation and Status

The components of the prototyped system are: the Player, the Recorder, the Session Directory (fig. 5), the RTSP client and server, the QoSFilter. As one can see from fig. 5, the JSD GUI displays the on-going MBone sessions upon which the user can navigate or join.

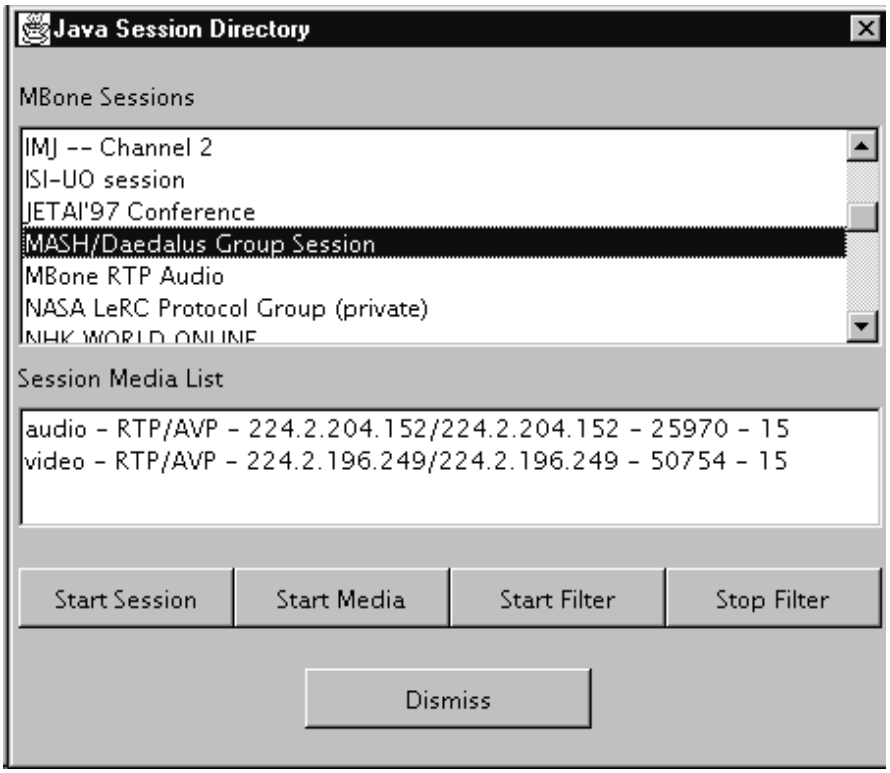

Figure 5 : Java Session Directory.

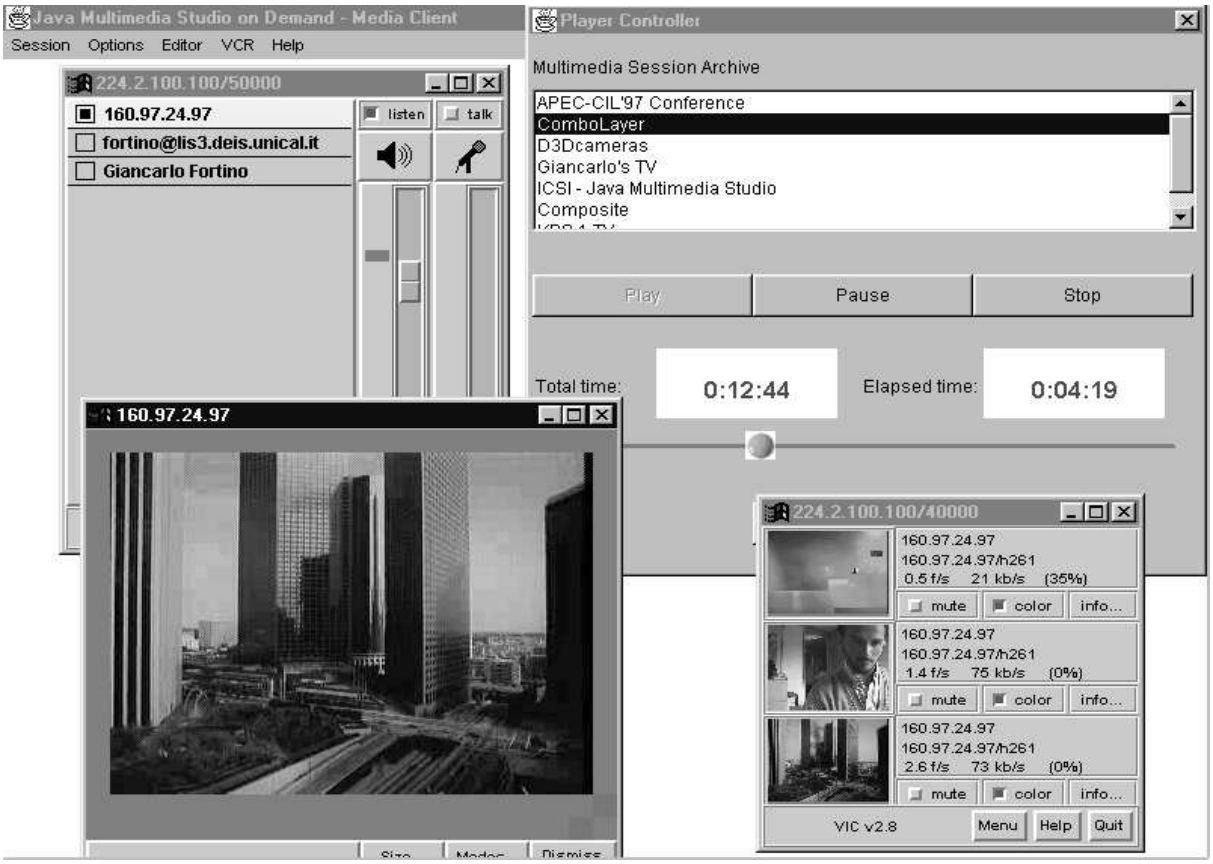

Figure 6 : Media Client “in action”.

Currently, the media client is implemented as both a standalone application and a helper application (see fig. 6) within a Web browser. The main advantage to invoking the client as a helper application, instead of a Java applet, is the that it can open multicast channels on behalf of the user. Besides, it is under development a whiteboard based on a reliable multicast protocol (LRMP [15]) that allows the members of a playback session to bookmark specific instants within a session and to collaborate by exchanging information.

Fig. 6 portrays a multimedia session with four synchronised streams (three video and an audio) controlled by the Player Controller through the commands *start*, *pause*, *stop* and a *time slider*. VIC and VAT are the media presentation tools.

5.CONCLUSIONS

The development of multimedia systems over Internet is a well-known complex task. It has strong QoS demands whose fulfilment require bandwidth control and real-time behaviour. However, the communication infrastructure remains basically out of control of the application. In addition, the use of standard OS features, namely over-killing and pre-emptive priority-based scheduling doesn't allow a fine level of timing control.

This paper proposes an approach based on actors and Java which integrates the achievement of the application requirements with a customisable scheduling. The actor notion is light-weight and relies on atomic message processing. Actors are used to define a framework of reusable media components which facilitate the configuration and deployment of multimedia systems based on network bindings, real-time protocols, media objects and media players.

The actor framework is under experimentation in the implementation of a Java Multimedia Studio on Demand system, which supports recording, playback and editing of multimedia sessions over MBone.

Current efforts are geared toward

- completing JMSoD prototype
- formalising the definition and the editing of media managers for the description of multimedia session timing and synchronisation constraints
- formalising actor-based multimedia systems by high-level Petri nets [21] in order to allow verification activities as a support to the design phase.

6. ACKNOWLEDGEMENTS

Work carried out under the financial support of the ICSI and the Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) in the framework of the Project "Methodologies and Tools of High Performance Systems for Distributed Applications".

7. REFERENCES

[1] Agha G., Actors : "*A model for concurrent computation in distributed systems*", MIT Press, 1986.

[2] Albanese A., Blomer J., Edmonds J., Luby M., "Priority Encoding Transmission", TR-94-039, ICSI Technical Report, 1994.

[3] Agha G., "Abstracting interaction patterns : a programming paradigm for open distributed systems", *Formal Methods for Open Object-based Distributed Systems'96*, Vol. 1, Najm E. And Stefani J. B. (eds), Chapman & Hall.

[4] Beraldi R., Nigro L. and Pupo F., "Actors and virtual time: an experience using Time Warp, timed Petri nets and cellular networks", *2nd IFIP Workshop on Formal Methods for Open Object-based Distributed Systems*, Canterbury (UK), 21-23 July 1997, Proceedings by Chapman & Hall, pp. 123-138.

[5] Fortino G., Nigro L., "QoS centred Java and actor based framework for real/virtual teleconferences", *Proc. of SCS EuroMedia 98*, Leicester (UK), Jan 4-6, 1998, 129-133.

[6] Fortino G., Grimaldi D., Nigro L., "Multicast control of Mobile Measurement Systems based on JAF", *Proc of IEEE IMTC/98*, S. Paul, Minnesota, May 1998, pp. 108-114.

[7] Fortino, G,. "Java Multimedia Studio v1.0", TR-97-043, ICSI Technical Report, November 1997.

[8] Handley M., Jacobson V., "SDP: Session Description Protocol", IETF, RFC 2327, April 1998..

[9] Handley M., "SAP: Session Announcement Protocol", Internet Draft, IETF, Multiparty Multimedia Session Control Working Group, Work in progress.

[10] Holfelder W., "Interactive remote recording and playback of multicast videoconferences", *IDMS'97*.

[11] Kirk B., Nigro L., and Pupo F., "Using real time constraints for modularisation." *Springer-Verlag*, LNCS 1204, 236-251, 1997.

[12] Kouvelas I., Hardman V., and Watson A., "Lip Synchronisation for use over the Internet: Analysis and Implementation", *Proc. of IEEE Globecom'96*, Nov., London UK.

[13] Kumar V., "*Mbone: Interactive multimedia on the Internet*", New Riders Publishing, 1996.

[14] Java Docum., http://java.sun.com/docs/

[15] Liao T., "Light-weight Reliable Multicast Protocol", version 1, 1997. http://www.inria.fr/webcanal.

[16] Malpani R. and Rowe L.A., "Floor Control for Large-Scale MBone Seminars", *Proc. of The Fifth Annual ACM Int. Multimedia Conf.*, Nov. 1997.

[17] McCanne S., et al., "Toward a common infrastructure for multimedia networking-middleware", *Proc. of 7th Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97)*, St. Louis, Missouri, May.

[18] Nahrstedt, K., Smith J., "The QoS broker." *IEEE Multimedia Magazine*, **2**(1), pp. 53-67, 1995

[19] Nahrstedt K., Smith J., "New algorithms for admission control and scheduling to support multimedia feedback remote control applications", *IEEE Int. Conf. On Multimedia Systems*, Hiroshima, Japan, June 1996.

[20] Nigro L., Pupo F., "A modular approach to real-time programming using actors and Java", *Proc. of 22nd IFAC/IFIP Workshop on Real-Time Programming*, Elsevier, Sept. 15-17, Lyon (Fr), pp. 83-88, 1997.

[21] Nigro L., Pupo F., "Using Design/CPN for the schedulability analysis of actors systems with timing constraints", *Proc. of Workshop on the Practical use of Coloured Petri Nets and Design/CPN*, Univ. of Aarhus (Denmark), 10-12 June, 1998.

[22] Parnes P., & others, "mMOD: the multicast Media-on-Demand system",

http://www.cdt.luth.se/~peppar/progs/mMOD/.

[23] Pazandak P., Srivastava J., "Multi-User Interactive Presentation Environments on the Internet: An Overview of DAMSEL and It's Implementation", *IEEE Conference On Multimedia Computing and Systems* (*IEEE MMS '96*), June, Hiroshima, Japan.

[24] Raman S., Shuett A., "On-demand Remote Playback", http://www-mash.cs.berkeley.edu/

[25] Ren S., Venkatasubramanian N., Agha G., "Formalising multimedia QoS constraints using actors", *2nd IFIP Workshop on Formal Methods for Open Object-based Distributed Systems*, Canterbury (UK), 21-23 July 1997, Proceedings by Chapman & Hall, pp. 139-153, 1997.

[26] Schulzrinne H., Casner S., Frederick R., Jacobson V., "RTP: a Transport Protocol for Real-Time Applications", IETF RFC1889, January 1996.

[27] Schulzrinne H., Rao A., Lanphier R., "Real Time Streaming Protocol (RTSP)", IETF RFC 2326, April 1998.

[28] Steinmetz R., Nahrstedt K., "*Multimedia: computing, communications and applications*", Prentice-Hall, 1995.