A Distributed Multimedia Toolbox

Hans Scholten, Pierre G. Jansen
University of Twente, Department of Computer Science,
POB 217, 7500 AE Enschede, the Netherlands
email: scholten@cs.utwente.nl

Abstract

Emphasis of our research lies on the application of real-time multimedia technology: tele-teaching, tele-conferencing and collaborative work. To support this research we need a real-time environment that supports rapid prototyping of distributed multimedia applications. Because other systems were not suited for our purpose, or did not fit the underlying framework of operating system and network, we decided to build our own environment, rather then using an available one and changing and extending it.

The resulting toolkit has the following features:

- *Network transparent multimedia services,*
- *Temporal access commands for continuous media,*
- *Quality of service management for all services,*
- *Low level inter media and spatial synchronization,*
- *Dynamic management of application topology.*

We found these features adequate for specifying, modeling and implementing distributed multimedia applications as is illustrated in the paper.

1. Introduction

Emphasis of our research lies on the application of real-time multimedia technology: tele-teaching, tele-conferencing and collaborative work. To support this research we need a real-time environment that supports rapid prototyping of distributed multimedia applications. Because other systems were not suited for our purpose, or did not fit the underlying framework of operating system and network, we decided to build our own environment, rather then using an available one and changing and extending it.

In section 2 we describe some work done on distributed multimedia systems. Section 3 gives the model used for the toolkit and section 4 it's architecture. An application of the toolkit is shown in section 5.

2. Related work

Substantial work is and has been done on distributed multimedia systems. Examples are found in SUMO [CBPS94], MMS [HIS93, HIS94], The Touring Machine [AKR92, Bel93] and MASI [BDFT]. In these systems, quality of service is an important issue. In the MASI project, everything depends on quality of service. In the SUMO project, all objects of the system have a quality of service specification. To provide transparent distributed multimedia most systems use object abstractions. Objects can be run anywhere on a network, and location dependent issues are handled by name and location servers. Grouping of objects is important to offer a consistent interface to groups of related objects. SUMO has flows to abstract a group of sequential-related connections, MSS has a group object to group virtual connections. These groups can be used for both parallel and serial grouping of objects.

3. Toolkit model

The multimedia toolkit consists of three types of objects:

- Actors
- End points on actors
- Methods at end points

3.1. Actors

An actor is the basic entity of activity. An actor is the only distributed object that actually does something. Other (user) objects in the system that are not actors cannot do anything. Producing, consuming, processing, controlling, generating, etc. are done by actors. Actors processing media can be compared with SUMO's devices [CBPS94] or with MSS's virtual devices [HIS94, HP93].

0-8186-8073-3/97 $10.00 © 1997 IEEE

3.2. End points

Actors must be able to communicate with other actors. This communication is done by means of end point to end point(s) communication in a location independent way. An end point is the only way for an actor to communicate with other actors. Here we can identify two kinds of communication: control communication and data communication. This duality results directly in two kinds of end points: control points and data points.

An actor can have several control points and several data points. Actors that produce or consume media data are called media drivers; they drive or process at least one medium stream.

3.3. Methods

In order to make an actor do something; one has to issue a command to this actor. Issuing commands is done by calling a method of a particular control point of the actor. This resembles the object-oriented way of message passing, as can be seen, for example, in MSS's —CORBA's— distributed objects. In some way methods also resemble handlers in SUMO —Chorus—.

Methods must be seen as dynamic primitives offered by a control point. Here, dynamic means that an actor can attach at run time a (internal, C) primitive to a control point, thereby making it a method. A method can also be detached by the actor who owns it. Methods are means to execute primitives that are not necessarily fixed or known at compile time or start-up time of client and server.

4. Architecture

4.1. Introduction

Actors, end points and methods together form the basic building blocks of our toolkit. With these, we can make a general framework for distributed multimedia handling. To provide the desired multimedia functionality several system actors are defined.

These basic multimedia capabilities include default temporal access control (TAC) [LG93] and QoS control through default methods of control points.

4.2. Media drivers

A media driver is a logical device driver that processes data of one or more media. They are abstractions for devices like cameras, speakers or media storage drives. This functionality is also seen in SUMO's devices and MSS's virtual devices.

Applications can request media driver services by issuing appropriate methods, after which control points, data points and connections are created. A media driver can get commands, like Start or SetQoS, and use other media drivers to realize the correct behavior of these commands. A media driver can also give feedback to other actors to monitor its behavior. Because media drivers are objects that have to be synchronized with each other, they must be capable of synchronizing.

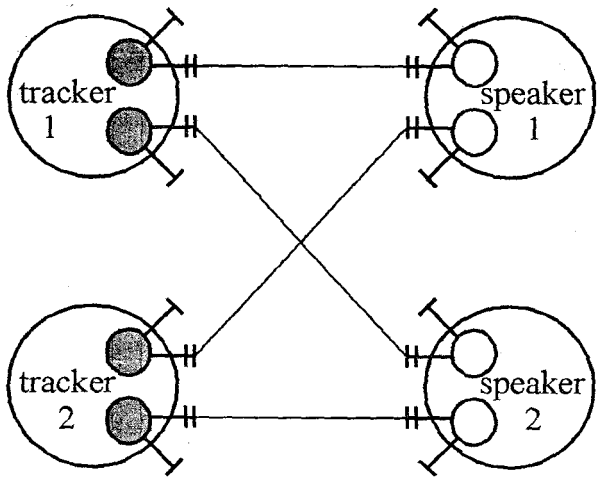
The MTP realizes the actual transport of bulk data by using network protocols, e.g., UDP or IP, shared buffers, or direct access of hardware, like DMA.

A media driver must have the following features in order to handle multimedia.

- *MMSPoint.* A standard control point that offers default and standard control over an actor. For example, methods like Open, Close, ReportOn, GetCPointList and Destroy can be issued here. This end point is very important, especially during the initialization phase of an application, because at that time initialization, configuration and tuning of different actors is needed. The MMSPoint also offers methods for getting information about the actor, e.g., about methods, control points, data points, feedback, etc.
- *Application.* A media driver is an application on its own, consisting of two parts: a basic actor part which causes the application to live in the toolkit's world; and an application specific part which has to realize the specific behavior of the application. The latter part is called the application.
- *Interface.* A control point/data point pair, shortly denoted as end point pair, is the bridge between its application and other actors transporting data or controlling this transport. Control is possible via the control point of this end point pair by issuing default control methods attached to it. These default control methods, the majority of them typically needed for multimedia systems, can be categorized as follows:
  - TAC. TAC is handled by the following methods: Start, Stop, Shoot, Prime, Flush, Goto, Where, SetSpeed and GetSpeed.
  - QoS. For QoS control two methods are needed: GetQoS and SetQoS.
  - Others. Methods to make an end point capable of handling state information requests, agreement between two connected data points, event based and periodic feedback.

With this multimedia package an actor is capable of generating, transporting and presenting media units in a way that abstracts from QoS – transportation, buffering timing and real-time behavior – and basic multimedia control like TAC.

FIGURE 1: AUDIO DATA TRANSPORT TOPOLOGY



4.3. System services

A media driver capable of handling multimedia does not make an application. Most multimedia applications consist of several actors all acting according a specific role and all interacting with each other. In order to abstract from these roles and interactions, and to control and synchronize actors —or group them—, the toolkit offers several system services.

These system services include a name server and a synchronization.

5. Applications

This chapter describes an application implemented with the toolkit. It is used to outline the capabilities of the multimedia toolkit , the way to use them and to test the functionality of the system. This exercise has shown that a consistent API is created, transparent distribution is provided, and multimedia transport and synchronized presentation are possible.

5.1. General Information

Before anything will run, a name server must be present in the system. Then the actors needed to run the application should be started. Once they are running, an application is created by connecting the actors, using a script file, shell commands or by running executable code.

Four phases can be distinguished when building an application. Firstly a data transport topology is initiated. Secondly, the required system services must be added to the application, i.e., logical and parallel grouping of connections and data streams, by using flows and synchronizers. Thirdly, the required quality of service should be specified. This can be done at the highest level of an application, as the quality of service specifications will automatically be distributed through the application. And, fourthly, the generation and presentation of media is controlled by using temporal access commands.

Currently a command line interface is provided for controlling applications at runtime. This may be completed with graphical interfaces in the future.

5.2. Audio Application

This example shows an application that uses a tracker and a speaker actor. A tracker actor generates music with up to four different voices, i.e. four different instruments can be played at the same time in parallel. One datapoint transmits one music channel which carries one of these four voices. A speaker actor receives audio samples at its data points, one channel per data point. The contents of the channels are mixed and played together on a local loudspeaker. The application expects all actors to be running. The audio application opens four source end point pairs and four sink end point pairs. Each tracker sends two channels of music out of four channels. Tracker 1 sends channels 1 and 3, tracker 2 sends channels 2 and 4. Each speaker actor receives one channel from tracker 1 and one from tracker 2. Music played at speaker 1 must be synchronized with music played at speaker 2. Figure 1 shows the data transmission topology for this application.

The following script constructs this topology for the application.

```
#
# Start tracker script
# ====================
#
```

```
# Open two end-point pairs at trackers
#
tracker1 1,2 Open
tracker2 1,2 Open
#
# Open two end-point pairs at speakers
#
speaker1 1,2 Open
speaker2 1,2 Open
#
# Create connections
#
connection 1 Open "con1 tracker1 1 \
speaker1 1"
connection 1 Open "con2 tracker1 2 \
speaker2 1"
connection 1 Open "con3 tracker2 1 \
speaker1 2"
connection 1 Open "con4 tracker2 2 \
speaker2 2"
#
# Open a synchronizer
#
synchronizer 1 Open "sync1"
#
# Add connections to the bundle
#synchronized by
# control-point 2 of the
#synchronization actor
#
synchronizer 2 AddSyncObj "sync1 con1"
synchronizer 2 AddSyncObj "sync1 con2"
synchronizer 2 AddSyncObj "sync1 con3"
synchronizer 2 AddSyncObj "sync1 con4"
#
# Set the QoS and real-time properties
#
synchronizer 2 SetQoS "MNDLY 0 MXDLY \
100 SKEW 100 PRD 10 MXJTR 100"
#
# End tracker script
# ==================
```

At this stage the topology of the application is set and ready to process data. Tests with this simple application have shown that synchronization based on global clocks results in acceptable skew values. Depending on the network load and CPU load stereo sound can be achieved. Currently we have been testing with up to 16 connections and 4 distributed sources and 4 distributed sinks. These tests also show that it may be advisable to integrate the flow actor, synchronization actor and connection actor into one system service actor, thus causing less network load.

6. Conclusions

We started this project to gain experience in designing and specifying real-time distributed multimedia applications. Based on a extensive literature study a multimedia toolkit is designed and built, thus offering a distributed multimedia platform for our experiments.

The toolkit has the following features:

- Network transparent multimedia services,
- Temporal access commands for continuous media,
- Quality of service management for all services,
- Low level inter media and spatial synchronization,
- Dynamic management of application topology.

We found these features adequate for specifying, modeling and implementing distributed multimedia applications. Anomalies, like a rather large latency from source to destination were due to heavy network load caused by other users of our network string. Although only initial tests are executed, we expect the multimedia toolkit to perform better with our new ATM based network.

7. References

[AKR92] Mauricio Arango, Michael Kramer, and Steven L. Rohall: "Enhancing the touring machine api to support integrated digital transport", Network and Operating System Support for Digital Audio and Video, 176-182, Springer Verlag, Berlin Heidelberg, 1992

[BDFT95] L. Besse, L. Dairaine, L. Fedaoui and W. Tawbi: "Towards an architecture for distributed multimedia applications support", In IEEE Multimedia, 1994

[Bel93] Bellcore: "Touring machine system", Communications of the ACM, 36(1), January 1993

[CBPS94] G. Coulson, G.S. Blair, P.Robin, and D. Shepherd: "Supporting continuous media applications in a microkernel environment", Technical Report MPG 94 16, University of Lancaster: Multimedia Projects Group, 1994, ftp://ftp.comp.lancs.ac.uk/pub/mpg/MPG 94 16.ps.Z

[CGHea] B.A. Coan, G. Gopal, G. Herman, and et al: "The Touring Machine System (Ver. 3): An Open Distributed Platform for Information Networking Applications", Bellcore

[GHV92] G. Gopal, G. Herman, and M. Vecchi: "The touring machine project: Toward a public network platform for multimedia applications", Proceedings of the 8th International Conference on Software Engineering for Telecommunications Systems and Services, March 1992

[HIS94] HP, IBM, and Sunsoft: "Multimedia Systems, chapter 9: Middleware Systems Services Architecture", 1994

[HIS93] HP, IBM and Sunsoft: "Multimedia System Services Version 1.0", 1993

[LG93] T.D.C. Little and A. Ghafoor: "Interval Based Conceptual Models for Time Dependent Multimedia Data", 1993

[MAH93] V. Mak, M. Arango, and T. Hickey: "The Applications Programming Interface to the Touring Machine", Bellcore, 1993

[Mil89a] David L. Mills: "Internet time synchronization: the Network Time Protocol", October 1989.

[Mil89b] David L. Mills: "Network Time Protocol (Version 2), Specification and implementation", September 1989