2nd CaberNet Plenary Workshop, Dublin, 23-25 March 1994

Programming Abstractions for Handling Multimedia Data in *CINEMA*

Tobias Helbig, Kurt Rothermel
University of Stuttgart
Institute of Parallel and Distributed High-Performance Systems
Breitwiesenstr. 20-22
D-70565 Stuttgart
Germany

Email: {helbig, rothermel}@informatik.uni-stuttgart.de

1 Introduction

An efficient and effective development of distributed multimedia applications is only possible if appropriate development platforms are provided. Those platforms, often referred to as middleware, sit on top of operating systems and provide various functions supporting distributed multimedia applications, including those for communication, synchronization and resource management. The *CINEMA* (Configurable INtEgrated Multimedia Architecture) system, which is under development at IPVR, falls into this system category. *CINEMA* focuses on communication and synchronization aspects of multimedia applications. It is able to convey and synchronize multimedia streams of arbitrary complexity: streams may have multiple sources and sinks and may comprise substreams of different types. Furthermore, the “path” connecting a source with a sink may involve any number of (end) systems. Therefore, in *CINEMA* the notion of end-to-end is used in a much broader sense than in the transport layer, where end-to-end characterizes a communication relationship between “directly” connected (end) systems. In *CINEMA*, end-to-end aspects are defined between the sources and sinks of a multimedia stream, where a stream may pass any number of intermediate end systems before reaching its sinks and thus may involve multiple transport connections.

CINEMA is built on top of a standard operating system and provides powerful programming abstractions, which simplify the development of distributed multimedia applications. Applications can be composed out of (possibly already existing) building blocks, so-called components, more complex components can be build out of existing ones, and components can be linked to arbitrary complex topologies. In order to convey multimedia applications, sessions with a certain quality of service can be set up, and multimedia flows can be controlled by media clocks.

The concept of a media clock is also used to specify synchronization requirements between related streams. All aspects of resource management, configuration management and where and how synchronization is done are hidden from the *CINEMA* user.

The remainder of the extended abstract is structured as follows. First, we introduce the basic abstractions of *CINEMA* and describe how they are used to build a multimedia application. Then, we show how to describe temporal and synchronization relations and how to control the transmission and presentation of data units. Finally, we give a short overview over the functional architecture of the *CINEMA* run time system and conclude with a brief summary.

2 Basic Abstractions of *CINEMA*

A multimedia system consists of hardware and software devices like speakers, microphones or one or more processors. Devices are mapped to components. A component is the abstraction that provides the multimedia processing functionality. It is equipped with a set of generic interfaces that enables the *CINEMA* run time system to control components independently of their actual functionality and implementation. Components send and receive data units of multimedia data streams via typed ports. It is distinguished between input and output ports. According to the input and output ports a component possesses it represents a source of a data stream, a filter, mixer or sink. There are no restrictions regarding the number of ports a component may have.

Components are interconnected via links. A link connects output and input ports of the same type and is used to describe communication paths between components. By that, the “topology” of an application is defined much the same way as in the REX system [MKSD90].

A sessions consists of the part of the topology an application wants to be treated as a single unit. The session is defined by a set of sources and sinks (intermediate components are found out internally) that are arbitrarily interconnected either directly or indirectly via intermediate components. Quality of service parameters are associated with sessions. When a session is created, *CINEMA* instantiates the involved components, establishes communication paths and reserves the resources (CPU, bandwidth, buffer) required to ensure the specified quality of service. So, in *CINEMA* a session is the unit of resource allocation.

Multimedia data is conveyed as flows, which are controlled by media clocks. Data units of flows are created at one or more sources. Each data unit has a timestamp that characterizes its temporal properties. Data units may pass filter or mixer components where they are processed or merged.

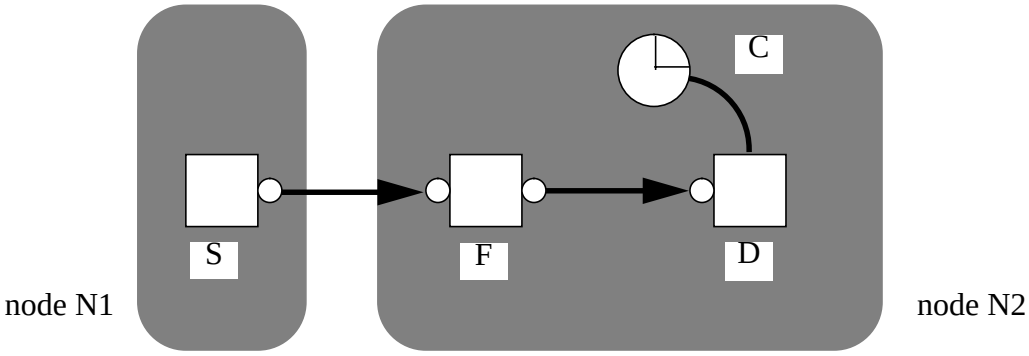
Finally, they are played out at sinks. Media clocks are associated with sources and sinks where they provide the interface to control flows. This is done by setting parameters of media clocks, by starting, stopping or changing the speed or direction of the advancing of the clock's time.

To control a flow, an application first defines the initial values of clocks which correspond to the timestamps of data units that are to be played out. Then, it sets rates with which the time of clocks changes and the stepwidth between consecutive values. These settings determine temporal properties of flows. The actual transmission of data units is started or stopped by starting or stopping media clocks since the advancing of time of media clocks coincides with the production, processing and presentation of timestamped data units. Clocks provide the means to selectively start or stop parts of flows (e.g. switch off a specific source or sink) as well as starting whole flows at once by manipulating groups of clocks.

At sinks, the settings of media clocks in conjunction with the timestamp of each data unit determine its play out time. The actual time of the sink's media clock corresponds to the timestamp of the data unit currently being played out. This provides an simple way to express the application's synchronization requirements. By correlating the values of the sink's media clocks, an application may specify the data units that are to be played out synchronously.

3 Example for Creating a Session

The usage of the above mentioned abstractions is illustrated by the following example. Assume, there is a file server at node N1 that allows the retrieval of video V (full motion, full color). It is intended to display V at a workstation (node N2). Unfortunately, the screen is only able to display black and white images. Thus, a filter is needed that transforms the color video into a black and white one.



Before viewing the film the appropriate components have to be created or fetched from a library. After that the topology will be specified. A file retrieval component S is to be specified for being instantiated at node N1 and initialized with video V. Then, the black and white display component D has to be specified for being created at node N2. Finally, a filter F is defined without mentioning any specific location. It may be chosen by the *CINEMA* system. After that, the links have to be defined. The output port of S is connected to the input port of F. The output port of F is to be linked to the input port of D. When the whole topology is declared to be the session, it is instantiated by the *CINEMA* system.

4 Flow Activation and Synchronization

After having created the session that led to the instantiation of components at appropriate nodes, the establishment of data transmission channels and the reservation of resources, the temporal properties of the flow have to be specified. Let's assume the video was recorded with a rate of 25 frames per second. The timestamps are equal to the frame numbers (first one is 1, increment is 1). It is intended to start the presentation with the 1000th frame and display the video with its original frame rate.

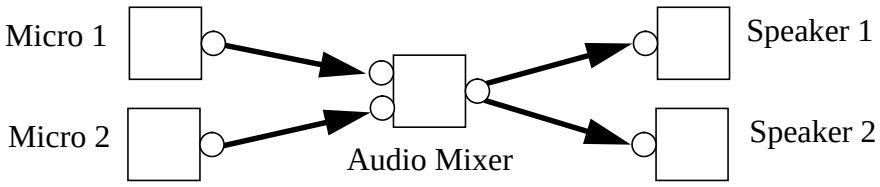
Thus, the parameters of the media clock being attached to the display are set to appropriate values: Its initial value is 1000. It "ticks" 25 times per second and each time changes its value by one. By issuing a start command at the media clock, the flow of data units from the file retrieval source S to the display D via the filter F is started instantly.

Changing the way data units are presented, is easily done by changing the parameters of the media clock. To jump to a specific frame in the video, the actual value of the clock is overwritten by the number of the frame. To slow down the presentation, either the clock's rate or its step-width is adapted. And viewing the video backwards requires the value of the clock to be decremented with each tick.

The following example illustrates how to control a more complex flow in a flexible way and how a synchronization relationship is defined.

As part of a teleconferencing tool, there is a mixer component that consumes data units from each connected input port, merges them into one data unit that is multicasted to the connected speaker components. To participate in the teleconference, an application connects its microphone and speaker component to the audio mixer and expands the existing session. It then spec-

ifies the temporal characteristics of the data units it will produce and consume (allowing the *CINEMA* system to perform consistency checks). By starting the media clock that is associated with the speaker, the application may start the flow of data units that originates at the mixer component. Consequently, it listens to the conversation of other participants without actually contributing to it as long as the clock of the microphone is not started. By starting the source's clock, the flow of data units from microphone to mixer is started, too. By issuing the start command to the microphone's and speaker's clocks at once, the application at once fully participates in the conference.



When it is intended to synchronize the multicasted output of the mixer component so that some or all participants of the conference receive the same data unit at the same time, this is specified by declaring a synchronization relationship between the speaker's clocks. Hence, the times of the media clocks are bound together, forcing the clocks to change their values synchronously. That leads to the synchronous output of data units.

5 The Functional Architecture of *CINEMA*

Internally, the *CINEMA* system is structured into four main building blocks. There is the configuration manager that configures components programmed by application programmers. It analyses specified topologies and decides where to place components. Furthermore, the configuration manager establishes the appropriate communication channels. The resource manager takes into account the application's quality of service needs. It manages the given resources of a computer system (CPU, network bandwidth, memory allocation, special devices) by reserving resources, monitoring given guarantees and graceful degrading the quality of service within specified margins in case of error or overload. The synchronization manager ensures the temporal requirements and synchronization relationships specified by an application and provides run time support for controlling the flow of data units. It instantiates the appropriate synchronization mechanisms depending on the application's requirements and provides flow control when transmitting data units. Finally, the event manager handles events and triggers the corre-

sponding actions. It is used internally to forward events and trigger actions, as well as externally to inform an application when a certain state of the presentation is reached.

6 Summary

We have briefly sketched the abstractions of the *CINEMA* system, which enables an application to build and control complex, distributed multimedia scenarios. *CINEMA* aims at closing the gap between the functionality provided by networks and operating systems and the demand for creating multimedia applications. Its emphasis therefore lies on end-to-end-issues that cover more than a single transport system hop and in supporting functional components with multiple input and output ports.

Thus, *CINEMA* differs in its approach from other research efforts currently being done. We neither try to expand the functionality provided by transport systems (OSI layer 4) by introducing grouping or synchronization issues at this level ([EDP92], [BCG+92]). Also, we do not want to tailor our approach to specific configurations such as being done by [AnHo91], [DNNR92], which essentially are extending window systems by multimedia functionality. Our work may be seen in the context of the goals defined by the IMA [Asso92] to support distributed multimedia applications by providing basic system services support.

Currently, the main abstractions and concepts of the *CINEMA* project and its architecture are defined. By now, we have an early prototype which supports simple application scenarios similarly to the examples above. Our future work is directed at refining our abstractions and developing protocols for resource reservation, stream control and synchronization as well as configuration management for arbitrary complex scenarios.

7 References

[AnHo91] David P. Anderson, George Homsy. Synchronization Policies and Mechanisms in a Continuous Media I/O Server. *Report No. UCB/CSD 91/617, Computer Sience Division (EECS), University of California, Berkeley, CA*, 2 1991.

[Asso92] Interactive Multimedia Association. Request for Technology: Multimedia System Services, Version 2.0, 11 1992.

[BCG+92] Gordon Blair, Geoff Coulson, Francisco Garcia, David Hutchinson, Doug Shepherd. Towards new Transport Services to Support Distributed Multimedia

Applications. In *4th IEEE ComSoc International Workshop on Multimedia Communications*, 4 1992.

[DNNR92] Roger B. Dannenberg, Tom Neuendorffer, Joseph M. Newcomer, Dean Rubine. Tactus: Toolkit-Level Support for Synchronized Interactive Multimedia. In *3nd International Workshop on Network and Operating System Support for Digital Audio and Video*, 11 1992.

[EDP92] Julio Escobar, Debra Deutsch, Craig Partridge. Flow Synchronization Protocol. In *IEEE Global Communications Conference*, 12 1992.

[MKSD90] Jeff Magee, Jeff Kramer, Morris Sloman, Naranker Dulay. An Overview of the REX Software Architecture. In *2nd IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, 10 1990.