Fifth ACM SIGOPS European Workshop, 21.-23.09.1992, Le Mont Saint-Michel, France

A System Software Structure for Distributed Multimedia Systems

Ralf Guido Herrtwich
Lars Wolf

IBM European Networking Center
Tiergartenstr. 8
D-6900 Heidelberg 1

rgh@dhdibm1.bitnet
lwolf@dhdibm1.bitnet

Digital audio and video data are different from traditional media in their time-criticalness and high bandwidth requirements. These requirements and the fact that typical multimedia applications perform only few operations on the continuous-media data suggest the use of new techniques for data handling in distributed, integrated, digital multimedia systems. This paper proposes a system software structure which encapsulates the processing of continuous-media data into stream handlers of a real-time environment. This environment is controlled by traditional non-real-time functions for resource, buffer and continuous-media data stream management. On top of this system, distributed multimedia applications can be built.

1 Introduction

Digital audio and video data are different from traditional media in their time-criticalness and high bandwidth requirements. Typical audiovisual data streams encoded with DVI or MPEG technology result in up to 1.5 Mbits of data every second; many applications such as video conferencing will require several of these streams to be handled within a single system. Each frame within a typical video stream needs to be processed within 33 ms to avoid backlog; rates for processing audio samples are usually higher although there is less data to handle.

Even fast computer workstations have problems with taking care of the requirements of continuous-media data when this data follows the same path as traditional data. These limitations will not go away any time soon with increased system performance: To compete with the consumer electronics domain, additional performance will be needed to improve the quality of multimedia presentation – in particular for digital compressed video. Therefore, alternate architectures for handling multimedia data are needed.

In typical multimedia applications, no or little processing of audio and video is required: The data is obtained from a source (microphone, camera, disk, network, etc.) and has to be forwarded to a sink (speaker, display, disk, network, etc.). The multimedia requirements are satisfied best when data takes the shortest possible path through the system. Preferably data is copied directly from adapter to adapter. The application then merely sets the correct switches for the data flow by connecting sources to sinks; it never obtains the actual data as in the traditional case.

This architecture of low-level data streaming is in line with proposals for using additional new busses for audio and video transfer within a computer. It also enables a switch-based rather than bus-based data transfer architecture which more and more authors suggest for multimedia computers (e.g., [1, 2]). From an application perspective, the user interface of such a system appears much like computer-supported telephony systems such as IBM's CallPath [3].

At the IBM European Networking Center in Heidelberg we implement a software system which is based on these ideas. HeiCoRe, the *Heidelberg Continuous-Media Realm*, provides low-level data streaming for arbitrary multimedia applications. It is developed for IBM RS/6000 workstations under AIX Version 3.2 as part of a complete future AIX multimedia platform.

2 Stream Handling Level

HeiCoRe (Figure 1) is based on top of device drivers. Above HeiCoRe exists the application support layer HeiMAT, the *Heidelberg Multimedia Application Toolkit,* which shields applications from low-level details of HeiCoRe. HeiMAT provides control functions which reduce the effort to create distributed multimedia applications. In particular, HeiMAT takes care of establishing continuous-media data flows across several systems by accessing different underlying HeiCoRe modules. HeiTS, the *Heidelberg Transport System,* is embedded into HeiCoRe and permits the exchange of continuous-media data with end-to-end performance guarantees. HeiTS uses communication protocols which support multimedia requirements: ST-II [4] in the network layer and HeiTP [5] in the transport layer.

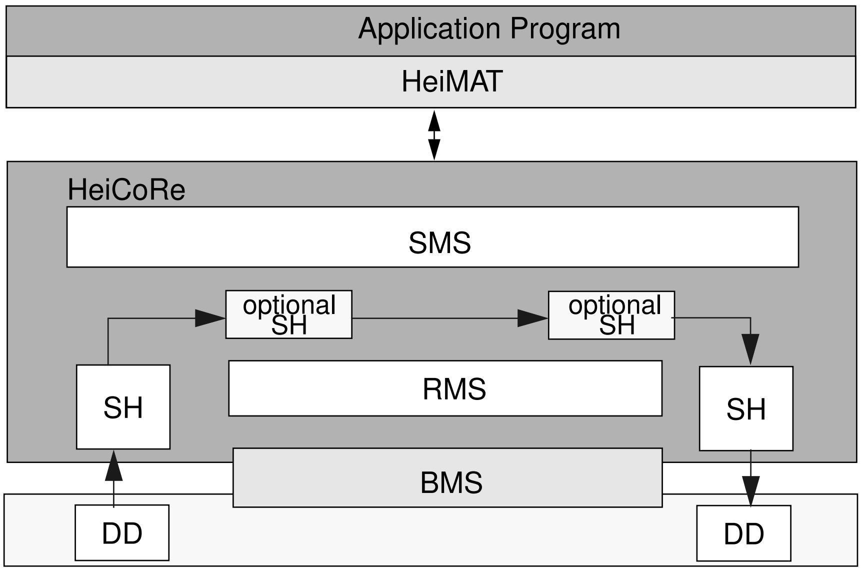

Figure 1: Overview of HeiCoRe.

The time-criticalness of continuous-media data requires the introduction of real-time techniques: resource scheduling has to take into account timing aspects instead of following traditional fairness policies only [6]. HeiCoRe provides the real-time environment to handle continuous-media data. It schedules the execution of data-handling functions through a rate-monotonic scheduler [7]. Rate-monotonic scheduling is well-suited for continuous-media data because of the regularly recurring data units. Deadline scheduling would be an alternative which potentially leads to higher resource utilization, but requires individual scheduling decisions for each data unit.

The real-time environment is controlled from non-real-time functions, e.g., the establishment of communication connections at the start of a stream has not to obey timing requirements, but the data processing for established connections has to. Applications determine the processing paths which are needed for their data and control the devices and paths; the execution of these time-critical functions is performed within HeiCoRe. Thus, applications usually have no direct contact with the continuous-media data – only if an application needs the data, the data will be delivered to the application but loses all time-characteristics.

To support the high volume data handling in HeiCoRe, special care is needed for data movement. The data of a continuous-media stream enters the system through an adapter. Usually, the device driver of this adapter has to copy the data into main memory before the data can be used. To avoid the overhead of copying data once again later, the same *Buffer Management Subsystem* (BMS) will be used in the device driver and in HeiCoRe, extending and generalizing the common *mbuf* scheme [8]. The BMS is designed to also work with shared memory adapters or even systems where the adapter can copy the data directly to another adapter.

The control part of HeiCoRe consists of the *Resource Management Subsystem* (RMS) and the *Stream Management Subsystem* (SMS). The RMS is responsible for reserving capacity for continuous-media data handling on scarce system resources [9, 10]. The SMS offers functions for the creation, modification, connection, control and destruction of *stream handlers* (SH) which perform the processing of continuous-media data streams. SHs make up the data handling part of HeiCoRe and exist in the real-time environment only. Their functions are dynamically linked to HeiCoRe.

Typical SHs are associated with I/O devices (e.g., the display or the network). Filters are SHs which modify the continuous-media stream without an I/O operation. An example of a filter is an audio mixer. It is even possible to generate data fully or in part in a SH (e.g., in computer animation) or to consume data in a SH (e.g., in voice or image recognition). The consumption of continuous-media data in a SH leads to discrete data instead of continuous-media data. If this data has to be presented to a user (e.g., movie subtitles), we have to obey timing requirements, otherwise (e.g., the data will be stored on a disk), the data can leave the real-time environment.

Each SH provides *endpoints* through which the continuous-media data flows. Data units enter a SH via one or multiple input endpoints and leave the SH through one or multiple output endpoints. An output endpoint of a SH has to be connected to an input endpoint of the next SH. To distinguish SH and network connections, the term *conduit* is used for SH connections.

In a typical multimedia system, the data comes from a source and enters an input endpoint of a SH. Between SHs, the data flows from output endpoints to input endpoints of other SHs. Finally, the data leaves the system through a sink. Because a SH may have more than one input and one output endpoint, the connection of SHs is not always a simple queue. All the SHs which participate in the processing of a continuous-media data stream form an acyclic, unidirectional graph, a *stream handler graph* (SHG) as illustrated in Figure 2.

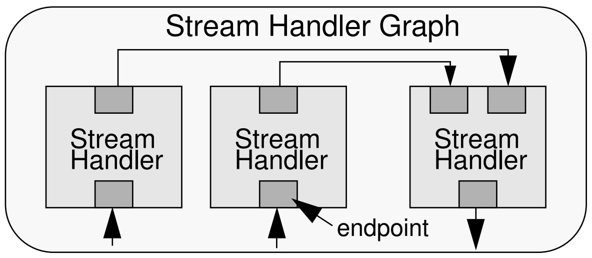

Figure 2: Stream handler graph consisting of three stream handlers.

The connection of endpoints is recorded in data structures. The delivery of data from an output endpoint to the next input endpoint is performed via a function call. This way, we have a chain of upcalls [11] from input SHs to output SHs and avoid the context switches which would be necessary with the use of a server-model [12]. This approach is similar to UNIX System V STREAMS [13, 14], but STREAMS is part of the operating system kernel and offers fewer mechanisms to support other structures than simple queues.

The functions of SHs are executed by threads. If a SHG is a simple chain of SHs, it is obvious that one thread can execute all SH functions in this SHG. If the SHs build a more complex structure than a simple chain, we have several choices. In the case that the SHG contains a mixing SH (which combines multiple streams into fewer streams) each incoming stream may have its own thread. All threads but one (or *n* for *n* output endpoints) finish their work through storing the data in a queue. The remaining thread pulls the data out of the queue and performs later processing steps. SHs which split one input stream into multiple output streams may have one thread for each output endpoint. Combinations of this methods are possible, e.g., one thread serves all the SHs connected to the output endpoints of one SH or one thread collects the data from all incoming streams of a SHG.

Real-time scheduling cannot prevent drifts in the operation of multimedia I/O devices. To regulate such drifts a feedback flow-control mechanism is used. The thread which calls an input endpoint function of a SH carries status information to adjust the flow of data when it returns to the prior SH. Each SH has to decide if it is able to adjust the rate itself. Usually, a SH which is not at the incoming end of a stream lets the preceding SH deal with the situation. The incoming SH may inform the scheduler to modify its scheduling interval or may drop or duplicate some data, e.g., if the data is generated from a device like an audio adapter. Even a SH which gets its input from a network connection will normally inform its preceding SH – it sends a control message to its peer on the sending host to adjust the amount of data. The protocols used in HeiTS offer this possibility.

Two or more continuous-media streams which are related to each other may come from different SHs, e.g., the video and audio part of a movie. Often, these streams need to be synchronized. HeiCoRe, therefore, includes a synchronization mechanism similar to the “logical time systems” from [15].

3 Application Level

To use HeiCoRe in a distributed system requires to establish conduits on different hosts. In our architecture, these functions are located in HeiMAT, the Heidelberg Multimedia Application Toolkit, one layer above HeiCoRe. HeiMAT can be used in two fashions:

- HeiMAT-based modules can follow the client/server approach, making distribution visible to the application. This would imply that every multimedia application is written in a way that takes distribution into account. This is shown in Figure 3.
- HeiMAT-based modules can follow the network transparency approach as made famous by the X window system [16]. In this case, multimedia applications are unaware of being used in a network environment. This is shown in Figure 4.

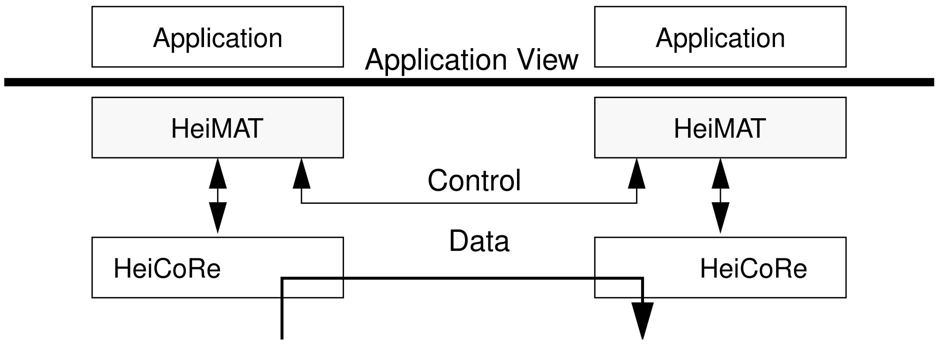

Figure 3: Distribution visible to application programmers (client/server).

Each approach has its justification: The client/server approach may lead to better performance and system response as it is possible to distinguish between local and remote execution of functions. For example, a stream stop operation could be performed locally, immediately freezing a video image on the screen. In the network-transparent case the stop function would always be sent to a remote application which may cause the video not to stop before one round-trip time.

On the other hand, with the network-transparent approach the application writer does not need to consider distribution when programming. One can come up with a solution similar to X to provide a universal I/O abstraction for multimedia. Unlike to X, however, data and control would follow different paths through the system. This is illustrated in Figure 4 where we see that each HeiMAT entity as part of an “M server” for audio and video I/O communicates with the HeiCoRe module on its local machine to affect the flow of local multimedia streams.

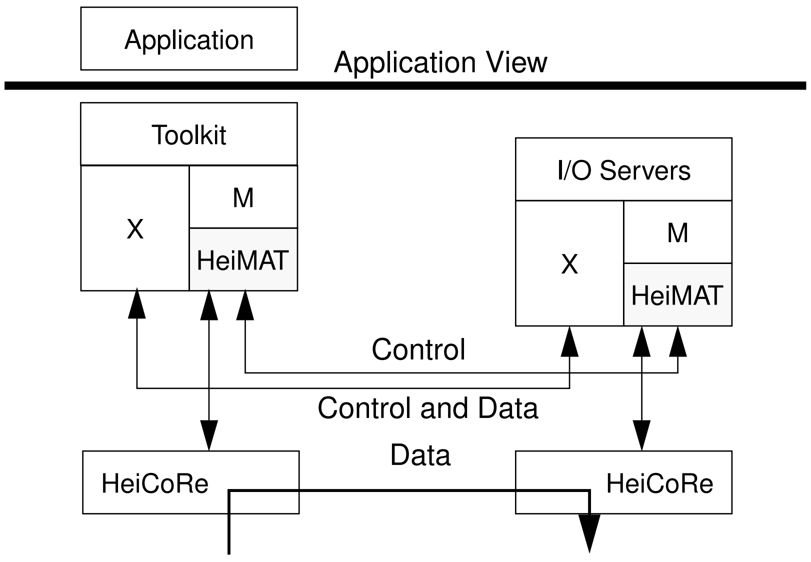

Figure 4: Distribution invisible to application programmers (network-transparent).

4 Conclusions

HeiCoRe is a platform which allows us – together with HeiMAT – to experiment with optimal handling of digital audio and video in a distributed system. It also demonstrates how to overcome shortcomings of traditional operating systems for continuous media. The current HeiCoRe version is designed for user space; the reasons for this were (1) easier implementation of the prototype system and (2) easier migration to a future microkernel operating system. The BMS works in both kernel and user space with shared memory so that device drivers can write directly into user memory. We also plan to include HeiCoRe in the actual AIX kernel to investigate how the performance of the system is affected.

References

[1] M. Hayter, D. McAuley: The Desk Area Network. ACM Operating Systems Review, Vol. 25, No. 4, pp. 14-21, Oct. 1991.

[2] G.G. Finn: An Integration of Network Communication with Workstation Architecture. ACM Computer Communication Review, Vol. 21, No.5, pp. 18-29, Oct. 1991.

[3] IBM CallPath DirectTalk/6000 Voice Processing System. IBM Program Number 5765-001.

[4] C. Topolcic (Ed.): Experimental Internet Stream Protocol, Version 2 (ST-II). Internet Request for Comments 1190, Network Working Group, October 1990.

[5] L. Delgrossi, C. Halstrick, R.G. Herrtwich, H. Stüttgen: HeiTP: A Transport Protocol for ST-II. Submitted to GLOBECOM' 92, Orlando, Florida, December 1992.

[6] J.G. Hanko, E.G. Kuerner, J.D. Northcutt, G.A. Wall: Workstation Support for Time-Critical Applications. Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, November 18-19, 1991, Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 1991.

[7] C.L. Liu, J.W. Layland: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM, Vol. 20, No. 1, 1973, pp. 46-61.

[8] Leffler, S.J., McKusick, M.K., Karels, M.J., Quarterman, J.S.: The Design and Implementation of the 4.3-BSD UNIX Operating System. Reading, Mass. et. al.: Addison-Wesley, 1989.

[9] D. Anderson, S.-Y. Tzou, R. Wahbe, R. Govindan, M. Andrews: Support for Continuous-Media in the DASH System. Proceedings of the Tenth International Conference on Distributed Computing Systems, Paris, May 1990.

[10] R. Nagarajan, C. Vogt: Guaranteed-Performance Transport of Multimedia Traffic over the Token Ring. IBM European Networking Center, Heidelberg, Germany, TR 43.9201, 1992.

[11] D.D. Clark: The Structuring of Systems Using Upcalls. Proceedings of the Tenth ACM Symposium on Operating Systems Principles, Washington, Operating Systems Review Special Issue, Vol. 19, No. 5, December 1985, pp. 171-180.

[12] L. Svobodova: Implementing OSI Systems. IEEE Journal on Selected Areas in Communications, Vol. 7, No. 7, September 1989, pp. 1115-1130.

[13] AT&T: UNIX System V, STREAMS programmer's guide, STREAMS Primer. Englewood Cliffs, New Jersey: Prentice Hall, 1987.

[14] D.M. Ritchie: A Stream Input-Output System. AT&T Bell Laboratories Technical Journal, Vol. 63, No. 8, October 1984, pp. 1897-1910.

[15] D. Anderson, G. Homsy: A Continuous-Media I/O Server and Its Synchronization Mechanism. IEEE Computer, Vol. 24, No. 10, October 1991.

[16] R.W. Scheffler, J. Gettys: The X Window System. ACM Trans. on Graphics, Vol. 5 No. 2, April 1986, pp. 79-109.