Design Issues on the Support of Tools and Media in Replayable Workspaces

Nelson R. Manohar and Atul Prakash
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109-2122 USA.
email: {nelsonr,aprakash}@eecs.umich.edu

Abstract

This paper presents flexible support for asynchronous sharing (later-time use) of computer-supported workspaces (CSWs), or simply replayable workspaces. Through the interactive replay of a previously recorded CSW session, it is possible to reuse valuable collaborative information. The session is represented through heterogeneous media streams each containing state and inputs to a CSW tool. We present here an architecture and tool/media interfaces for the support of tools as "plug-in" components of our replayable workspaces.

1. Introduction

Suppose that while interacting with applications on your desktop, you reach an interesting result. You are likely to document these results and procedures so maybe, *later on*, you can share these — results and procedures — with your colleagues. For example, consider the computer-supported workspace, (herein CSW), of a radiologist. The sharing, at a later time, of such CSW is valuable to radiologists (e.g., for intern training), to clinicians (e.g., for consultation and analysis), and to administrators (e.g., as active documentation). From a collaborative viewpoint, our goals *wrt* a CSW are to: (1) *reproduce intra-task content*, e.g., at a later time, review the steps taken that led to a diagnosis; (2) *review the task*, e.g., at a later time, browse/annotate the session; and (3) *reuse the results and procedures*, i.e., build upon the results contained in the workspace.

Our approach addresses the above goals by supporting the *asynchronous sharing* of interactive CSWs — through the use of **session objects** [14]. Such asynchronously-shared CSWs are referred herein as **replayable workspaces**[1].

Our long-term research focuses on the development of toolkits to support collaboration paradigms such as the asynchronous sharing of CSWs. This research is partially motivated by the needs of two projects at the University of Michigan: UARC [5] and the Medical Collab (MDC). Specifically, in the MDC, our goal is the asynchronous sharing of the CSW of a radiologist. To demonstrate our replayable workspace concepts, we used the University Hospital of Geneva's OSIRIS II tool[2]. Our prototype[3] represents a black box that extends replay-awareness to an Osiris session by coordinating audio, video, and window tools and their media. Snapshots of this sample session are shown in Figs. 1-4.

In this paper, we analyze architectural support needed to achieve flexible and modular replayable workspaces. The rest of this paper is organized as follows. First, we explain the problems that need to be addressed. Second, we discuss related work. Then, we describe a CSW model for flexible tool coordination. Similarly, we describe a media model for flexible media integration. Next, we analyze our media integration. Similarly, we analyze our tool coordination. Finally, we make our concluding remarks. In **Appendix A**, we specify the interfaces of our architecture.

[1]— in contrast to the term *shareable* workspaces used in synchronous collaborative systems, the term replayable highlights the asynchronous (later-time use) nature of our research.
[2]— at *http://expasy.hcuge.ch/www/UIN/osiris.html.*
[3]— at *http://www.eecs.umich.edu/~nelsonr/systems.html.*

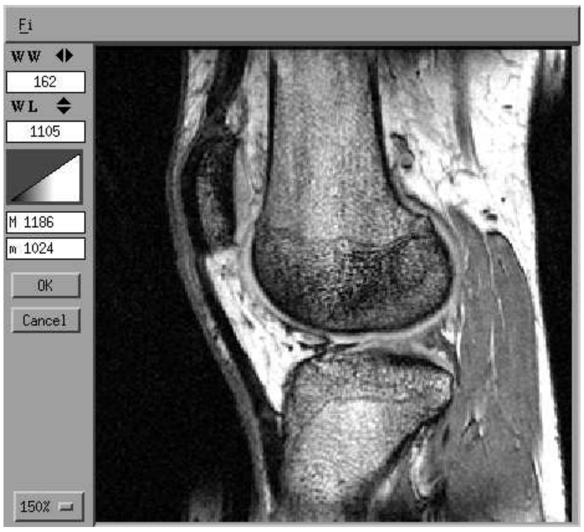

Figure 1. The prototype CSW consists of audio, video, and X-windows multiplexer tools. A session manager coordinates these tools. Here, an x-ray is loaded into OSIRIS II, which runs under the X-multiplexer tool. The prototype allows capture and replay of any such session, annotated with either (or both) audio and video (not shown).

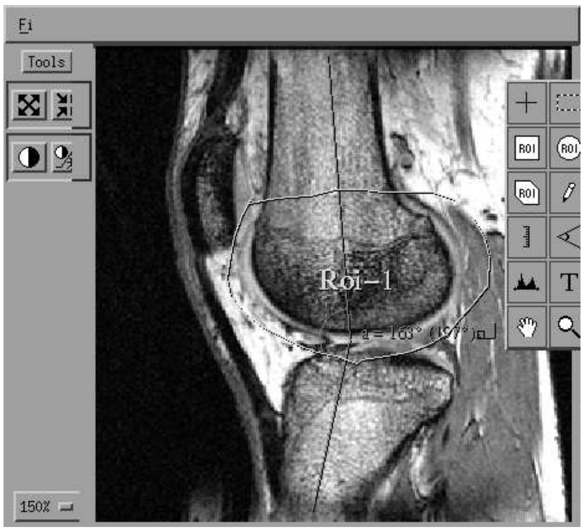

Figure 3. Analyzing the x-ray. A tool palette is accessed (upper right side) and its angle measurement tool is selected. The tool is now being used to measure the angle at which the knee joint is bent (lines on the center). A measurement of 163 degrees can be found (lower right of Roi-1).

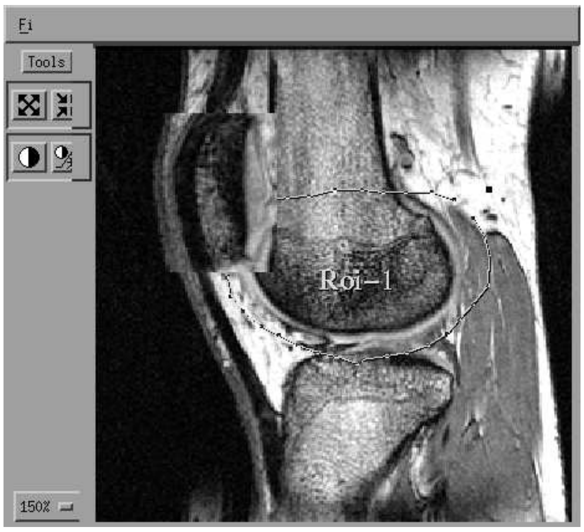

Figure 2. Close-up viewing of the x-ray. Using OSIRIS, a region of interest (Roi-1, middle) is created and a magnifying glass tool is being used to examine details (upper-left of Roi-1).

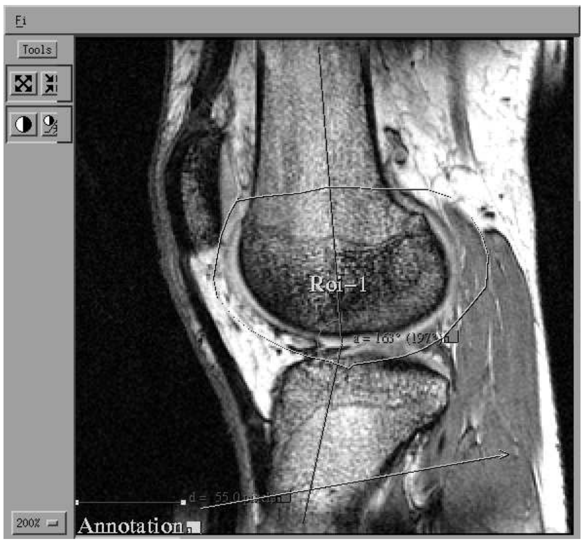

Figure 4. More analysis over the x-ray. An annotation was made (bottom left), the image was scaled from 150% to 200%, and a measurement was taken.

Temporal Media Stream		Degree of Freedom *df*
Key	Description	example *df* resource
LTS	global timing	n/a
A^c	continuous audio	n/a
A	discrete audio	silences
V	video	frame rate
W	window stream	inter-event delay
D	application data	inter-event delay
C	command stream	inter-event delay

Table 1. Typical streams found in CSWs.

`LDU Charact.`	A^c	A	V	W	D	C
Definition	*frame*	*phrase*	*frame*	*event*	*datum*	*command*
Continuity	Y	N	Y	N	N	N
Statefulness	N	N	Y/N	Y	Y	Y
Periodicity	Y	N	Y	N	Y/N	N
Time-Variant	N	N	N/Y	Y	Y/N	Y

Table 2. Media characteristics of tms events.

2. Problem statement

In this section, we outline fundamental issues on the asynchronous use of CSWs. A CSW is composed of multiple tools. A *tool* is a user-level process that renders services to the CSW. A *service* is a user-oriented function provided to the CSW, Usage over time of a tool service is modeled by a temporal media stream (*tms*). In particular, Table 1 describes some *tms* of interest to us. These *tms* have a range of heterogeneous characteristics.[4]

For example, our prototype CSW is composed of three tools (and their services): (1) an audio tool (records and replays audio), (2) a video tool (records and replays video), (3) an X-Windows multiplexer (re-distributes the X display stream). Although each tool has some temporal media awareness, the aggregation of these tools (their CSW), lacks integration of both tool services as well as their media. To support later-time use of CSWs, we must address the following problems in temporal-awareness:

tool coordination: The re-execution of CSWs requires temporal coordination of tool services. In particular, we desire tool coordination mechanisms that are: (1) independent of tool and media characteristics and (2) flexible enough so as to selectively "plug-in", replace, or manipulate a tool without affecting other tools.

media integration: The re-execution of CSWs requires integration of heterogeneous *tms* (see Table 2). In particular, we desire media integration mechanisms that are: (1) independent of media characteristics and (2) flexible enough to meet the range of requirements of our heterogeneous media.

In addressing the tool coordination and media integration problems, we must consider how services are to be mapped to tools. There are two basic approaches.

In the first approach, the services of multiple tools are integrated into a monolithic-application — using carefully orchestrated, cooperating threads. Our previous prototype followed this approach [13]. This approach delivers complete control over tool coordination and media integration since, at the thread level, fine-grained scheduling and synchronization control is possible. Unfortunately, this low-level control also limits the flexibility of the resulting infrastructure.

In the second approach, each tool remains a user-level process in the replayable workspace. However, the coordination of tool services is now handled by an intermediary process — in our case, a session manager, (as in *publish-and-subscribe* systems such as ToolTalk). In this paper, we describe the architecture and protocols that are used between session manager and the tool services to achieve both tool coordination and media integration.

[4]— such as (1) logical data unit (**LDU**) definition, (2) LDU continuity, (3) LDU statefulness, (4) LDU periodicity, and (5) time variance on LDU re-execution.

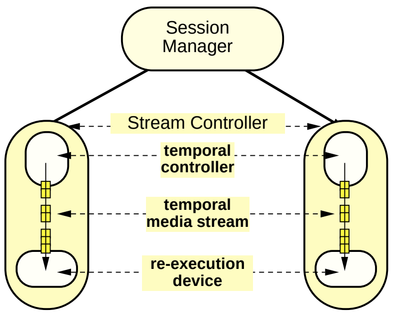

Figure 5. The architecture of replayable CSWs.

3. Related work

X-Window-aware systems, such as [1, 6], in principle, allow playback of unmodified applications. However, these systems lack media integration and interacting with the underlying CSW is not possible.

In media authoring, synchronization relationships are explicit, (i.e., manually *crafted* via scripts or flowcharts so as to balance media processing). In our artifact generation (i.e., the capture of interactive CSW sessions), synchronization relationships are implicit, (i.e., a user-transparent process). Although our research then seems similar to interactive multimedia presentation research[5] [3, 4], we focus on fine-grain integration of heterogeneous media streams as opposed to orchestration of heterogeneous document parts.

Transportable active objects are also found in Java. Our architecture represents a framework that will allow such program capsules to replay platform-independent CSWs. We are currently pursuing this research.

Steinmetz characterized media-oriented QoS (e.g., synchronization and continuity) requirements in [23]. In general, application-level media integration can not meet the full QoS requirements spectrum. Through reliance on a re-executable input model, our media-oriented QoS requirements relaxes into a range such as needed by voice-annotated window events — i.e., a less QoS-restricted range ($[250 - 1000]ms$) [23], feasible for application-level support [13].

Media servers rely on a tighly-coupled media/synchronization server [7, 20] for fine-grained synchronization and processing of pre-selected continuous media. Such coupling can be inflexible — through media-integration that is dependent on the media types being handled. Our approach removes awareness of media types from media integration by (1) delegating media processing to stream controllers and (2) providing various tradeoff points (between synchronization and continuity) for application-level media integration of a range heterogeneous tms.

The playback of stored media requires addressing three basic overheads: (1) fetch, (2) processing, and (3) presentation. Research on management of fetch overheads [19, 20] and scheduling overheads [10, 17] focuses on (low-level) system extensions to meet media requirements for continuous media. Our work focuses on application-level management and can be built on top of such extensions. Kernel-level management of additive overheads between competing processes is approached in [16]. One could visualize our session manager as a kernel process for support of replayable workspaces on future multimedia operating systems. Our research on abstract machines for media-independent tool coordination would then become central to support "plug-in" compliance of tools to this kernel extension.

Although a sporadic server can be used for integration of discrete/continuous media with hard guarantees [9, 22], this assumes that any discrete event can execute within the time bound provided. For some tms, (e.g., a command stream [8]), a feasible time bound for LDU re-execution may not be practical. We focus on application-level integration of heterogeneous media and show how it is designed to statistically support a range of media-oriented QoS requirements.

4. Architecture overview

A CSW is composed of multiple tools. A tool is incorporated into a replayable workspace by means of a user-level process, refer to as a *stream controller*. A stream controller extends *temporal-awareness* to a particular tool

[5]— e.g., stored media integration, interactive browsing.

Media Integration Description				**Synchronization Treatment**	
P	Policy	Integration Requirements	Illustration	rhs ahead lhs cond.	lhs ahead rhs cond.
P1	*laiseez faire*	rhs/lhs have large asynchrony tolerance or rhs/lhs exhibits negligible timing variability.	audio playback against a global timing track: $(LTS \leftarrow A^c)$.	do nothing	do nothing
P2	*rhs-waits*	lhs has strong continuity requirement. rhs exhibits negligible timing variability.	audio-annotated MIDI sequences: $(A \leftarrow M)$.	rhs waits for lhs report on synchronization event s_i.	do nothing
P4	*even-wait*	lhs/rhs subject to timing variability.	lip-synchronous video: $(A \leftarrow V)$.	same as for P2.	lhs waits for rhs report on s_i.

Table 3. Non-adaptive mechanisms for inter-stream synchronization. Based on the protocol assigned to (lhs, rhs) and the asynchrony condition, a "look up" yields the synchronization treatment assigned to (lhs, rhs).

service and its media. A user-level process, referred to as a session manager, coordinates these temporally-aware tools (stream controllers). Fig. 5 illustrates our replayable workspace architecture (see [15]). The session manager manipulates these stream controllers through simple "*VCR-like*" commands — regardless of the services provided by the tools. The stream controller abstraction has three components:

- *controller* — (delivers temporal-awareness to its CSW tool),
- *temporal media stream* — (herein *tms*, represents a temporal ordering of the data inputs to a tool and its services), and
- the *re-execution device* — (the underlying media device or the tool itself). The re-execution device is modeled as a consumer/producer of *tms* events.

To decouple the session manager from heterogeneous tools, a homogeneous interface is specified between the session manager and its tools. Stream controllers enforce a homogeneous interface to media-dependent tool services. For example, a request to "*playback a session*" gets broadcast to stream controllers as the abstract command: "*replay stream*".

To decouple the session manager from heterogeneous media handled by tools on its workspace, its interfaces are specified over a media-independent stream type. Media-independent commands are dispatched from the session manager to stream controllers. Stream controllers provide media-dependent execution of commands. The implementation of media-dependent functions is thus transparent to the session manager.

A flexible workspace also requires decoupling the logical representation of a session object from the physical representation of any of its temporal media streams (*tms*). A two level data model is used for our session objects. The *intra-tool data model* refers to the physical layout of a *tms*. This needs to, and thus is only known, to its stream controller. The *inter-tool data model* refers to the logical representation of a CSW session (meta-data). This needs to be agreed to by all streams controllers, so as to enforce homogeneous manipulations over a CSW session.

A temporal media stream models inputs to a tool as a temporally-ordered sequence composed of events and state(s). Events represent units of re-execution, and as such are defined by their re-execution device model. Events represent deltas applied over application state and may be statefully dependent on previous events, state, or both. State data provides the basis for a restartable process formulation that allows re-execution of events at an arbitrary time-index.

Finally, session objects are transportable objects, rooted as a file hierarchy of media directories [14]. The session manager remains unaware of stream-dependent details such as naming scheme, data locality, re-execution device, or layout of stream controller repositories. The data of a stream controller is currently represented using standard UNIX filesystem support (i.e., byte-stream files and directories). The session object is thus copied and moved as any other directory.

Media Integration Description				Synchronization Treatment	
P	Policy	Integration Requirements	Illustration	rhs ahead lhs cond.	lhs ahead rhs cond.
P3	*rhs-waits and adapt*	lhs has strong continuity requirement. rhs exibits timing variability. rhs has degree of freedom.	audio-annotated window events: $(A \leftarrow W)$	same as for P2 and if rhs tends to run-ahead, decrease rhs speed.	if rhs tends to fall-behind, increase rhs speed.
P5	*even-wait and rhs-adapt*	lhs/rhs subject to timing variability. lhs has strong continuity requirement. rhs has degree of freedom.	$(A^c \leftarrow VW)$ or $(LTS \leftarrow AVW)$.	same as for P3.	same as for P4 and if rhs tends to fall-behind, increase rhs speed.

Table 4. Adaptive mechanisms for inter-stream synchronization.

5. Media model

Our media integration model is based on **synchronization events** coupled with a *master/slave model* — both widely accepted in the synchronization literature [21]. Our contribution focuses on flexible, application-level support for heterogeneous media integration.

Based on individual tms tolerances and requirements, a relative tradeoff between *synchronization* and *continuity* can be associated with each master/slave relationship. Each continuity/synchronization tradeoff level is supported by a protocol and its associated *synchronization treatment*. When a stream controller falls "*out-of-synch*" *wrt* its master, several remedial treatments are possible. The various synchronization treatments supported are summarized in Table 3 (also in [13]). In the next section, will show how various treatments relate to continuity/synchronization tradeoff levels. Here, we describe the synchronization infrastructure that supports the various protocols of Table 3.

Our architecture uses the session manager as a centralized resource to address media integration between stream controllers. To ameliorate inter-process communication (IPC) overheads, the synchronization infrastructure uses flexible intervals coupled with relative scheduling to handle integration of asynchronous tms, as in [11, 13]. An interval is a sequence of consecutive tms events that is delimit at each end by a synchronization event. For example, the i^{th} such interval is $(s_i, e_1, \cdots e_n, s_{i+1})$.

The synchronization infrastructure is built around two general mechanisms:

scheduling mechanism: The scheduling of the temporal media streams is decomposed into scheduling intervals. A scheduling interval represents the *same time segment* across all streams controllers.

synchronization mechanism: An $s_i(lhs \leftarrow rhs)$ synchronization event, preserves relative timing between lhs and rhs scheduling intervals. In this notation, rhs synchronizes to lhs.

The temporal inter-media correspondence between scheduling intervals across multiple streams is maintained as follows. During the capture of a session, a synchronization event is posted at a well-defined endpoint of an lhs stream[6] and it is then inserted into its rhs streams[7]. This has two side-effects: (1) it creates a scheduling interval and (2) it establishes a synchronization relationship between the rhs and lhs media streams. The replay of a session is implemented by (1) chaining the playback of consecutive scheduling intervals (for each stream) and (2) synchronizing parallel progress of these inter-related scheduling intervals. The scheduling of a synchronization event $s_i(lhs \leftarrow rhs)$, causes the inter-stream asynchrony between lhs and rhs to be estimated. Based on (1) the protocol assigned to (lhs, rhs) and (2) the current asynchrony condition, a "look up" to Table 3, yields the synchronization treatment assigned to (lhs, rhs).

6. Flexible tool coordination

Next, we show how the tool coordination abstract machine supports the specification of the features of session objects. These primitives for tool coordination are specified in **Appendix A**. For brevity, we focus only on browsing features. Previous research in temporal access control (TAC) [2, 12, 18, 20], show modeling of "*Vcr-like*"

[6]— e.g., assuming $(A \leftarrow VW)$, at the end of an audio frame.
[7]— e.g., on both the window and video streams.

browsing features as temporal transformations over a logical time system. We show our browsing support to be independent of the tool and media characteristics. Throughout this discussion: (1) recall that a session object is composed of scheduling intervals, bounded by scheduling events $(s_1, ..., s_n)$ and (2) let str be the set of all stream controllers in a replayable workspace.

6.1. Replaying

The REPLAY() command enables the playback of a block of scheduling intervals. The following enables chained playback from the current scheduling interval (say s_k) to s_n: $\texttt{REPLAY}(str, s_n)$.

Internally, stream controllers implement this command as a "back-to-back" chaining of scheduling intervals. The REPLAY() command only *enables* playback, whereas the actual dispatching of scheduling intervals is controlled by the media integration mechanism. The media integration mechanism executes between scheduling intervals. A naive chaining implementation is likely to introduce intra-media discontinuities (such as media playback continuity glitches). However, our architecture empowers stream controllers to stream-dependent optimizations to compensate for this. For example, based on the strong sequential-access pattern of this paradigm, prefetching can be used to ameliorate this effect.

Finally, scheduling intervals also allow the session manager to support user interactivity during the replay of a sesion. For example, before the dispatch of the next scheduling interval, the session manager can (1) perform housekeeping (such as updating progress feedback) as well as (2) interact with the user (for example, pause the session).

6.2. Pausing and resuming a replay

A PAUSE() command requests stream controllers to stop playback at the end of the next feasible scheduling interval. However, stream controllers may stop at different scheduling intervals (say s_j^i). Consequently, the session manager determines a common scheduling interval s_j,[8] across its stream controllers. The session manager then requests stream controllers to playback up to the common scheduling interval s_j. The following models such interaction (pause/continue) over the playback of a session object:

$$\left\{ \begin{array}{l} \texttt{REPLAY}(str, s_n); \\ s_j = \texttt{PAUSE}(str); \texttt{RESUME}(str, s_j); \\ \texttt{REPLAY}(str, s_n); \end{array} \right\}$$

6.3. Browsing

Several forms of browsing are of interest to us.

- Dynamic browsing. To fast-replay a session, the session manager requests its stream controllers to increase their presentation rate. For example, fast-forward from s_i to s_j is modeled as:
$$\left\{ \begin{array}{l} s_i = \texttt{PAUSE}(str); \texttt{RESUME}(str, s_i); \\ \texttt{SET}-\texttt{SPEED}(str, speed); \\ \texttt{REPLAY}(str, s_j); \end{array} \right\}$$
- Static browsing. A stream controller is responsible for providing static browsing of its *tms*. For example, an audio tool can display waveforms whereas an application can display a temporal ordering of its data inputs. Browsing at s_i is modeled as:
$$\left\{ \begin{array}{l} s_i = \texttt{PAUSE}(str); \texttt{RESUME}(str, s_i); \\ \texttt{BROWSE}(str, s_i); \end{array} \right\}$$
- Random access to an arbitrary scheduling interval s_i. Session objects are (very) stateful objects. If state capture is feasible for a stream controller (e.g., as with object-oriented, open-interface classes like in Java), we can rely on periodical state snapshots so as to jump to the nearest snapshot before s_i and fast replay from there on. Alternatively, if state capture is infeasible, we can rely solely on fast replay. The FORWARD()

[8]— for example, $s_j = max_i(s_j^i)$.

command encapsulates such media awareness:

$$\left\{ \begin{array}{l} \texttt{FORWARD}(str, s_i; speed); \\ s_i = \texttt{PAUSE}(str); \texttt{RESUME}(str, s_i); \end{array} \right\}$$

6.4. Dubbing

Dubbing of a *tms* is a complex operation because of: (1) intra-media statefulness and (2) inter-media synchronization. To preserve statefulness, dubbing is restricted to well-defined boundaries only (i.e., state snapshots). To preserve synchronization, dubbing is restricted to *rhs* streams. To dub an *rhs* stream, the session manager requests: (1) the *rhs* stream controller to record while (2) the $(str - rhs)$ other stream controllers replay. Synchronization events are inserted into the *rhs* stream by the session manager on behalf of *lhs* (by means of the SYNC-INSERT() command, defined in **Appendix A**). The following dubs s_{i+1} through s_{j-1}:

$$\left\{ \begin{array}{l} s_i = \texttt{PAUSE}(str); \texttt{RESUME}(str, s_i); \\ \texttt{RECORD}(rhs); \\ \texttt{REPLAY}(str - rhs, s_{i+1}); \\ \texttt{SYNC} - \texttt{INSERT}(rhs, s_{i+1}); \\ \ldots \\ \texttt{REPLAY}(str - rhs, s_{j-1}); \\ \texttt{SYNC} - \texttt{INSERT}(rhs, s_{j-1}); \\ \texttt{END} - \texttt{OF} - \texttt{RECORD}(rhs); \\ s_j = \texttt{PAUSE}(str - rhs); \texttt{RESUME}(str - rhs); \end{array} \right\}$$

7. Flexible media integration

Temporal relationships between *tms* are specified through synchronization relationships. Multiple synchronization relationships may be needed to represent a CSW. However, not all synchronization relationships have the same media-oriented QoS requirements. This is particularly true for our heterogeneous media, which exhibits a range of tolerances on: (1) continuity between sequential scheduling intervals *and* (2) synchronization between parallel scheduling intervals. Thus, we are interested in exploiting such range of tolerances. Here, we present application-level brokerage of tradeoffs in media synchronization and continuity.

The decoupling of tool coordination from media processing introduces inter-process communication overheads into our media integration. In this section, we design application-level tradeoffs between synchronization and continuity, so as to ameliorate the impact of such overheads. To this end, we first explain how, during playback, the session manager estimates the inter-stream asynchrony between (lhs, rhs) processes. Then, to support latter analysis, we describe the generic processing steps taken to synchronize them. Last, we show how, given these conditions and overheads, we can design protocols to target various tradeoff levels between media synchronization and continuity.

7.1. Estimation of inter-media asynchrony

When each stream controller schedules a synchronization event s_i, a measurements report, SYNC-UPDATE(), is sent to the session manager. The i^{th} such report contains global timestamps for the *start* and *end* of the i^{th} synchronization event. This report is described in detail in **Appendix A**. The session manager uses the reports from a (rhs, lhs) pair to estimate the inter-stream asynchrony (ϵ_i) between them as follows.

1. **measurements collection:** SYNC-UPDATE() reports from *rhs* and *lhs* stream controllers are gathered. Suppose that *lhs*/*rhs* reports contain parameters (respectively):

$$lhs : (s_{i+1}, start_i^{lhs}, end_i^{lhs}) \quad (1)$$
$$rhs : (s_{j+1}, start_j^{rhs}, end_j^{rhs}) \quad (2)$$

2. **asynchrony estimation:** Two inter-stream asynchrony estimates are computed:

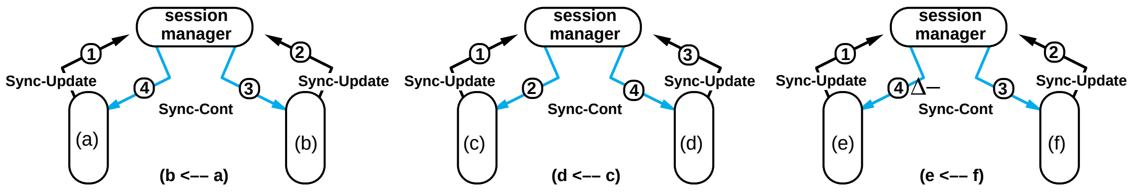

Figure 6. Synchronization treatments for handling of the (rhs ahead lhs) condition. The above CSW is modeled by three synchronization relationships: ($b \longleftarrow a$), ($d \longleftarrow c$), and ($f \longleftarrow e$). Media integration relies on a synchronization handshake between stream controllers. A stream controller pauses scheduling execution to request synchronization through a SYNC-UPDATE() report. Upon reception of a SYNC-CONT() message, it resumes execution. Synchronization treatments are applied by the session manager and are transparent to stream controllers. Each protocol allocates different effort to treatment and detection. Synchronization relationships can be assigned to different protocols (i.e., multiple protocols can coexist). For example, the above synchronization relationships rely on the following synchronization treatments: blocking (P2/P4), "laiseez faire" (P1), and adaptive blocking (P3/P5), respectively.

- just *before* the i^{th} synchronization event:

$$\epsilon_{start}(lhs, rhs) = start_i^{lhs} - start_j^{rhs} \tag{3}$$

- just *after* the i^{th} synchronization event (ϵ_i):

$$\epsilon_{end}(lhs, rhs) = end_i^{lhs} - end_j^{rhs} \tag{4}$$

3. **efficiency evaluation:** The efficiency of media integration is evaluated. For each rhs and lhs pair, the following (jitter) statistic is maintained:

$$\alpha_i = \epsilon_{end} - \epsilon_{start} \tag{5}$$

7.2. Synchronization handshake

This distributed algorithm performs a synchronization handshake between rhs and lhs stream controllers. The handshake uses two messages: the SYNC-UPDATE() report and the SYNC-CONT() command. SYNC-UPDATE() is used by a stream controller to report it has completed scheduling the i^{th} scheduling interval. SYNC-CONT() is used by the session manager to enable a stream controller to process its next ($i + 1^{th}$) scheduling interval. These messages are described in detail in **Appendix A**.

This synchronization handshake is used by all protocols. The implementation of the various synchronization treatments is encapsulated within the session manager and is transparent to stream controllers. For example, Fig. 6 shows a CSW modeled by three synchronization relationships, where each is assigned to a different protocol (i.e., multiple protocols can coexist). This results in a different synchronization treatment (blocking, *laissez faire*, and adaptive blocking, respectively), to be applied to, for example here, an rhs ahead lhs condition. Synchronization treatments can be differentiated only by the ordering of messages to/from the session manager for a given rhs/lhs pair. Next, we outline the generic steps taken in the handling of the rhs ahead of lhs condition.[9]

1. rhs **initiation**: rhs report reaches session manager:
   (a) rhs posts a SYNC-UPDATE() report.
   (b) rhs suspends scheduling — until it receives its SYNC-CONT() command.
2. **synchronization treatment**: Based on the synchronization treatment assigned to the rhs and lhs set, (see Table 3), the session manager may or may not impose a synchronization wait over rhs.

[9]— Without loss of generality (wlg), we use an rhs initiation example. For lhs to rhs, similar algorithms exist.

Figure 7. Session playback is scheduled in scheduling intervals. A scheduling interval relates a time segment across media streams. Overheads such as synchronization and message latencies affect the resulting scheduling interval. The scheduling interval can be used to tradeoff inter-media synchronization to media continuity. Here we illustrate overheads that result from using the P4 (2-Way)-blocking policy — in the worst case, results in a scheduling re-start at each scheduling interval.

3. lhs **initiation**: Independently, some time later, the lhs report reaches the session manager:
   (a) lhs posts a Sync-Update() report.
   (b) lhs suspends scheduling — until it receives its Sync-Cont() command.
4. rhs **and** lhs **termination**: The session manager terminates the synchronization handshake between rhs and lhs stream controllers,[10]:
   (a) Sync-Cont() command sent to lhs.
   (b) Sync-Cont() command sent to rhs.

7.3. Trading synchronization and continuity

Next, we analyze the impact of media integration over synchronization and continuity and show how we can use such knowledge to design flexible media integration. First, we characterize our synchronization overhead. Then, we analyze the impact of this overhead over media continuity so as to show how we can specify tradeoffs between synchronization and continuity.

7.3.1 Media integration overheads

Let T be the expected duration of a scheduling interval. Let $t_{latency}$ be the average inter-process communication latency. We will estimate the average time (t_{sync}), incurred by our synchronization handshake (see Fig. 7).

The synchronization handshake has four phases. The completion of each phase incurs in a corresponding time component in t_{sync}, being as follows:

- $t_{latency}$ — time cost for the rhs initiation phase.[11]
- $f(T)$ — average waiting time for the synchronization treatment given by the session manager to the reporting rhs stream controller.[12]
- $t_{latency}$ — time cost for the lhs initiation phase.
- $2 * t_{latency}$ — time for rhs/lhs termination phase.

Thus, our synchronization handshake requires time:

$$t_{sync} = 4 * t_{latency} + f(T) \tag{6}$$

[10]— If no assumptions are made about message delivery, an ordering policy may be specified over which stream controller (i.e., lhs or an rhs) is to be notified first.
[11]— Although assumes bounded delay, reliable Fifo delivery, our Ipc is limited to intra-workstation messages.
[12]— For example, under protocol P3, the average time $f(T)$ is close to $T/2$ [13].

We are also interest in the duration in real time (RT) of a scheduling interval (T^*), measured as:

$$T^* = RT(s_i) - RT(s_{i-1}) \tag{7}$$

but when blocking protocols are used, we approximate T^* as:

$$T^* \leftarrow T + t_{sync} \tag{8}$$

Finally, we characterize the ratio of synchronization overhead to scheduling time as the scheduling efficiency of the synchronization mechanism q:

$$q = \frac{t_{sync}}{T} = \frac{4 * t_{latency}}{T} + \frac{f(T)}{T} \tag{9}$$

To support application-level control, we rely on relatively large scheduling intervals T. In particular, in our case, $t_{latency}$ becomes negligible *wrt* T. Since $f(T)$ is a function of the protocol, T, and the load conditions [13], this results in relatively large $f(T)$ values. Consequently, eqs. (8) and (9) reduce to:

$$T^* \leftarrow T + f(T) \tag{10}$$

$$q \leftarrow \frac{f(T)}{T} \tag{11}$$

7.3.2 Impact over media-oriented QoS

Our media integration targets two QoS indicators: (1) intra-media QoS (e.g., media continuity) and (2) inter-media QoS (e.g., media synchronization). Media synchronization and continuity are competing processes[13] — it is possible to achieve good continuity and relatively poor synchronization as well as it is possible to achieve poor continuity and relatively good synchronization. Our media integration is designed to yield various tradeoff points (by trading off continuity to synchronization) to meet a range of heterogeneous media requirements. Next, we develop means to objectively compare these tradeoffs.

Our inter-media QoS is media-independent — it relates the quality of inter-stream synchronization to indicators such as: (1) the inter-stream asynchrony ϵ_i and (2) the asynchrony jitter α_i. Since the waiting time $f(T)$ depends on the inter-stream asynchrony ϵ_i, q also depends on ϵ_i. This is important because we will show that furthermore, intra-media continuity depends on q. Thus, we can then design synchronization protocols to *allocate treatment effort* over ϵ_i with various degrees of scheduling efficiency. Consequently, synchronization to continuity tradeoffs are specifiable.

Quality indicators for intra-media QoS are media-dependent. For continuous media, intra-media QoS qualifies media continuity, whereas for discrete media, it qualifies the smoothness of the scheduling of media events. However, we show that in either case, the corresponding quality indicator depends on q, as follows:

- **For discrete media**; let m_i be the ratio of schedule time (LTS) to real time (RT):

$$m_i = \frac{LTS(t_i) - LTS(t_{i-1})}{RT(t_i) - RT(t_{i-1})} \tag{12}$$

  The scheduling of discrete media can be adapted. For example, a resource such as inter-event delays (as for W) or silence (as for A) can be adapted to fit synchronization needs. Let w_i be the current adaptation effort over a scheduling interval, — (i.e., T is scheduled as w_iT). Evaluating m_i at $\Delta t = T$, we get:

$$m_i = \frac{T}{w_iT + f(T)} = \frac{1}{w_i + q} \tag{13}$$

  The first difference of m_i represents temporal jitter. When perturbations on the m_i time series are due only to noise, playback of the w_i-compensated scheduling intervals proceeds at a stable rate.

[13]— Both intra-media and inter-media QoS indicators can be either directly or indirectly affected by $f(T)$.

- **For continuous media**; let c_i be the timing departure from the expected duration of a media frame (c) to the real time elapsed on its playback:

$$c_i = \frac{c}{RT(c_i) - RT(c_{i-1})} \tag{14}$$

  Evaluating c_i at $T = c$, we get:

$$c_i = \frac{T}{T + f(T)} = \frac{1}{1+q} \tag{15}$$

  The first difference of c_i represents continuity jitter. When perturbations on the c_i time series are due only to noise, playback of the n scheduling intervals proceeds at a stable rate.

Thus, synchronization determines q and q determines continuity. Good continuity calls constant and small q. The efficiency metric q depends, primarily, on the synchronization effort $f(T)$ — (applied to the inter-stream asynchrony). We can design synchronization protocols to perform at various $f(T)$ values by allocating effort on detection and treatment of asynchrony conditions [13]. Consequently, we can obtain various tradeoff levels between synchronization and continuity.

8. Experience

We have preliminary experience on the use of the framework outlined in this paper in the context of the medical collaboratory project. A prototype is implemented in C and under Solaris 2.4.[14] The prototype workspace is composed of: (1) audio stream controller — based on soundtool; (2) video stream controller — based on Sun's xilcis; and (3) window stream controller — based on XMX (Brown University X-windows multiplexer tool)[15]. Stream controllers (tools) run as independent servers, coordinated by our session manager. Processes run in the same workstation and share non-realtime resources such as CPU, window server, and disk.

8.1. User Experience

Radiologists in our group made the following observations after using the system:

- Ability to make recordings between arbitrary time periods in an interactive session is very important.
- Being able to go back and edit a recording is important. It was important to radiologists to be able to record a polished version of the session.
- Synchronization of audio and pointing was important but radiologists were to used to speaking very precisely about regions of interest. So, occasional lack of close synchronization was not disruptive. Coarse grain synchronization (approximately within one second) appeared satisfactory.
- Availability on multiple OS platforms is important because of the use of various kinds of platforms in the hospital environment.

The design decisions regarding tool coordination aspects address the needs pointed out to us by the radiologists. Currently, we are focusing on JAVA applications for reasons of platform-independence and easier state capture. We plan to develop modular ways to extend full replay-awareness to Java applications that will allow for their co-existence in our framework.

[14]— at *http://www.eecs.umich.edu/˜nelsonr/systems.html.*
[15]— found at *http: //www.cs.brown.edu/software/xmx/v1.html.*

8.2. Media Integration

Given a certain set of synchronization treatments ensuring that q is small and constant is important, for both continuity and synchronization.

Our architecture uses loosely-coupled applications, in order to provide more flexibility in integrating various independent tools. Synchronization overheads due to inter-process communication between the session manager and the tools are higher than they would be in a monolithic application, such as that described in[13]. Those overheads basically depend on the value of $t_{latency}$ as compared to the value of T, the synchronization interval. The values of $t_{latency}$ on most workstations is variable but usually less than $10ms$. We have found that choosing T, the scheduling interval, to be between one to two seconds to be satisfactory for synchronizing audio and window events. Since $t_{latency}$ is small compared to T, it contributes little to the value of q or variations in q.

The results of the various synchronization protocols for one $\{rhs, lhs\}$ pair in a tightly-coupled application were reported in [13]. There, we showed that, it is important to choose the appropriate synchronization treatments between the streams, taking into account the nature of the streams. For instance, in the case of audio and window streams, it is better to make audio a master stream (for continuity) and window a slave stream (for synchronization).

Given the synchronization relationships between the streams, the use of adaptive protocols such as P3 or P5, as shown in [13] can reduce both the value and variations in asynchrony, and thus $f(T)$. Thus, use of adaptive protocols can help in reducing the value of q and the variations in q.

9. Conclusion

Our research on replayable workspaces represents an enabling framework for transportable active objects that allow re-execution of sessions with a CSW. In this paper, we presented support for asynchronous (*later-time*) use of CSWs. To support this new technology, we presented incremental contributions to a large body of related work. In particular, we addressed the later-time use and coordination of tool services and their media. We designed mechanisms for application-level control of tool coordination and media integration. Mechanisms were designed to be flexible by decoupling them from tool and media characteristics. Our tool coordination abstract machine was shown to be extensible. Our media integration was designed to meet various tradeoff levels between synchronization and continuity.

10. Acknowledgments

Our work has been supported in part by NSF Grant ECS-94-22701, by NSF Cooperative Agreement IRI-92-16848, and by the Rackham Merit Fellowship.

A. Inter-tool and inter-media interfaces

In this section, we specify the interfaces that support the tool coordination and media integration problems. By complying with these interfaces, a stream controller delivers temporal-awareness to services from its respective CSW tool to the collective replayable workspace.

To support tool coordination and media integration, every stream controller must comply with three inter-process interfaces *wrt* the session manager (Fig. 8) [15]:

functional interface (F): supports tool coordination — its primitives provide abstract machines that build the features of session objects.

synchronization interface (S,D): supports media integration — its primitives support fine-grained inter-media relationships between *tms*.

feedback interface (M): supports the evaluation of the efficiency of the above interfaces.

When a stream controller complies with our interface specifications, its corresponding CSW tool is said to be **replay-aware**. A replay-aware tool is modeled as a temporally-aware plug-in component to our replayable workspaces. Among other things, it is able to record and re-execute its services and media.

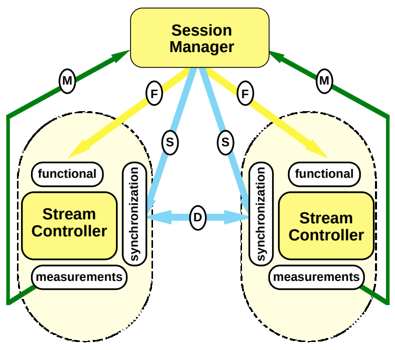

Figure 8. Interfaces of the architecture. The architecture uses a client – server model. The session manager coordinates multiple CSW tools — each managed by a stream controller.

A.1. The functional interface

These primitives support tool coordination. The session manager translates user requests into abstract commands, forwarded to stream controllers. These media-independent commands enforce a homogeneous interface to media-dependent services of heterogeneous stream controllers. These commands use the forward-channels $F()$.

SELECT($str, enable$**):** *enables* or *disables* stream controllers until further notification. The parameter $enable$ indicates whether or not str is to be enabled.

RECORD(str**):** a *nonblocking* call that enables stream capture at a receiving stream controller str.

END-OF-RECORD(str**):** disables str stream capture.

SAVE($str, basename$**):** causes str to create persistent representation(s) of its temporal media stream. The parameter $basename$ contains the path name to the storage hierarchy of the session object.

OPEN($str, basename$**):** causes str to load the persistent representations of its stream into memory. The parameter $basename$ contains the path name to the storage hierarchy of the session object being loaded.

REPLAY(str, s_i**):** enables str to chain-replay from the current scheduling interval up to the i^{th} scheduling interval, inclusive.

SET-SPEED($str, speed$**):** requests str an unconditional and absolute change of its LTS scheduling rate to the value specified by the parameter $speed$.

$s_j =$ **PAUSE(**str**):** requests str to pause its processing at the end of its next feasible scheduling interval (say s_j). The scheduling interval s_j is returned to the session manager.

RESUME(str, s_j**):** causes str to resume processing but only up to its s_j scheduling interval.

BROWSE(str, s_i**):** causes str to launch a content browser of its temporal media. The content browser uses the time position implied by s_i to display the static contents of str temporal media stream.

ABORT(str**):** causes str to abort its current operation.

A.2. The feedback/measurements interface

These feedback primitives allow the session manager to evaluate the efficiency of media integration and tool coordination. These messages use the back-channels $B()$.

DID-INIT(str**):** reports that str has initialized.

CMD-COMPLETED(str**):** reports *explicitly* that str has completed processing of its last command.

SYNC-ISSUE(lhs, s_i**):** requests (during capture of a session) a synchronization relationship between this lhs to its rhs stream controllers to be established at the synchronization event s_i.

SYNC-UPDATE($str, i+1, start_i, end_i, \omega_i$**):** triggered by the scheduling of a synchronization event s_{i+1}, causes str to collect scheduling measurements $(start_i, end_i)$. The times $start_i$ and end_i represent CSW-wide timestamps for the start and end, respectively, of the processing of the *previous* synchronization event (s_i) at str. The value ω_i is used by adaptive protocols and represents the compensation factor currently in use by str's LTS. For non-adaptive protocols, it is set to 1.

A.3. The synchronization interface

These primitives provide support needed by a variety of synchronization treatments — (see Table 3).

SYNC-INSERT(lhs, rhs, s_i): inserts a synchronization event s_i into rhs *wrt* to an lhs stream controller — thus establishing a relative synchronization relationship at s_i, i.e., $s_i(lhs \leftarrow rhs)$.

SYNC-CONT($str, s_i, \epsilon, \Delta$): terminates the synchronization handshake with str (i.e., releases synchronization block and resumes scheduling). The parameter ϵ specifies an estimate of the inter-stream asynchrony between rhs and lhs stream controllers. The parameter Δ is used by adaptive protocols and specifies a recommended adaptation effort. For non-adaptive protocols, it is set to 0. A positive Δ suggests a rate increase whereas a negative Δ suggests a decrease.

References

[1] H. Abdel-Wahab, S. Guan, and J. Nievergelt. Shared workspaces for group collaboration: An experiment using Internet and Unix inter-process communication. *IEEE Comm. Magazine*, pages 10–16, November 1988.

[2] D. Anderson and R. Kuivila. A system for music performance. *ACM Transactions on Computer Systems*, 8(1):56–82, February 1990.

[3] D. Bulterman, G. van Rossum, and R. van Liere. A structure for transportable, dynamic multimedia documents. In *Proc. of the Summer 1991 USENIX Conference*, pages 137–154, Nashville, TN, USA., June 1991.

[4] M. Cecelia-Buchanan and P. Zellweger. Scheduling multimedia documents using temporal constraints. In *Proc. of the 3rd Int'l Workshop on Network and Operating System Support for Digital Audio and Video*, pages 237–249, La Jolla, CA, USA, November 1992.

[5] C. Clauer, J. Kelly, T. Rosenberg, C. Rasmussen, P. Stauning, E. Friis-Christensen, R. Niciejewski, T. Killeen, S. Mende, Y. Zambre, T. Weymouth, A. Prakash, G. O. S.E. McDaniel, T. Finholt, and D. Atkins. A new project to support scientific collaboration electronically. *EOS Transactions on American Geophysical Union*, 75, June 28 1994.

[6] E. Craighill, R. Lang, M. Fong, and K. Skinner. CECED: A system for informal multimedia collaboration. In *Proc. of ACM Multimedia '93*, pages 436–446, CA, USA, August 1993.

[7] R. Dannenberg, T. Neuendorffer, J. Newcomer, D. Rubine, and D. Anderson. Tactus: toolkit-level support for synchronized interactive multimedia. *Multimedia Systems*, 1(1):77–86, 1 1993.

[8] J. Herlocker and J. Konstan. Commands as media: design and implementation of a command stream. In *Proc. of ACM Multimedia '95*, pages 155–166, San Francisco, CA, November 1995.

[9] K. Jeffay. On latency management in time-shared operating systems. In *Proc. of the 11th IEEE Workshop on Real-Time Operating Systems and Software*, pages 86–90, Seattle, WA, USA, May 1994.

[10] K. Jeffay, D. Stone, and F. Donelson. Kernel support for live digital audio and video. *Computer Communications*, 15(6):388–395, July 1992.

[11] L. Li, A. Karmouch, and N. Georganas. Multimedia teleorchestra with independent sources: Part 1 — temporal modeling of collaborative multimedia scenarios. *Multimedia Systems*, 1(2):143–153, 1994.

[12] T. Little. A framework for synchronous delivery of time-dependent multimedia systems. *Multimedia Systems*, 1(1):87–94, 1993.

[13] N. Manohar and A. Prakash. Dealing with synchronization and timing variability in the playback of interactive session recordings. In *Proc. of ACM Multimedia '95*, pages 45–56, San Francisco, CA, November 1995.

[14] N. Manohar and A. Prakash. The Session Capture and Replay Paradigm for Asynchronous Collaboration. In *Proc. of European Conference on Computer Supported Cooperative Work (ECSCW)'95*, pages 161–177, Stockholm, Sweden, September 1995.

[15] N. Manohar and A. Prakash. A flexible architecture for heterogeneous media integration on replayable workspaces. In *Proc. Third IEEE Int'l Conf on Multimedia Computing and Systems, to appear*, Hiroshima, Japan, June 1996.

[16] C. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves for Multimedia Operating Systems. In *Proceedings of the IEEE ICMCS*, May 1994.

[17] J. Nakajima, M. Yazaki, and H. Matsumoto. Multimedia/realtime extensions for the Mach operating system. In *Proc. of the Summer USENIX Conference*, pages 183–196, Nashville, TN, USA, Summer 1991.

[18] S. Ramanathan and P. V. Rangan. Continuous media synchronization in distributed multimedia systems. In *Proc. of the 3rd Int'l Workshop on Network and Operating System Support for Digital Audio and Video*, pages 328–335, La Jolla, CA, USA, November 1992.

[19] P. V. Rangan and H. M. Vin. Designing file systems for digital video and audio. *ACM Transactions of Computer Systems*, 18(2):197–222, June 1992.

[20] L. Rowe and B. Smith. A Continuous Media Player. In *Proc. of the 3rd Int'l Workshop on Network and Operating System Support for Digital Audio and Video*, pages 376–386, La Jolla, CA, USA, November 1992.

[21] R. Steinmetz. Synchronization properties in multimedia systems. *IEEE Journal of Selected Areas of Communication*, 8(3):401–411, April 1990.

[22] R. Steinmetz. Analyzing the multimedia operating system. *IEEE MultiMedia*, 2(1):68–84, Spring 1995.

[23] R. Steinmetz and K. Nahrstedt. *Chapter 15: Synchronization*, pages 585–595. Prentice Hall, 1995.