J-SAC/8/3//34089

Ralf Steinmetz:
[Ste90-2] *Synchronization Properties in Multimedia Systems*. In: *IEEE Journal on Selected Areas in Communication*, vol. 8, no. 3, p. 401--412, April 1990.

Synchronization Properties in Multimedia Systems

Ralf Steinmetz

Reprinted from
IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS
Vol. 8, No. 3, April 1990

Synchronization Properties in Multimedia Systems

RALF STEINMETZ, MEMBER, IEEE

***Abstract*—"*Multimedia*" refers to the integrated generation, representation, proeessing, storage, and dissemination of independent machine processable information expressed in multiple time dependent and time independent media such as data, graphics, drawings, voice, audio, and video. The availability of the required technologies now and in the future stimulates research and development of products in this area.**

"*Synchronization*" assures a temporal order of events. Synchronization mechanisms are a well-studied topic in the area of operating systems, parallel programming languages, and database technology.

This paper addresses the characteristics of synchronization mechanisms desirable for central and distributed multimedia systems. The concept of "multimedia objects" as components of an object based model for a multimedia system is introduced. The essential new synchronization requirement is *restricted blocking* together with synchronization features covering *real-time* aspects. Existing synchronization mechanisms can be altered or new ones defined to meet these requirements.

I. Introduction

THE developments and achievements in the area of

- video technology (such as displays, cameras),
- media dependent compression techniques (e.g., algorithms for 64 kbit/s codecs of moving video signals or the RELP algorithm for voice coding [26]),
- high speed networks (such as optical fibers, broadband ISDN, FDDI [22], QPSX [19]),
- mass storage technology (e.g., optical memories [4], [16]),
- processing power (32-bit data bus and address bus as standard, 64-bit wide in the future) and large memories (DRAM's) [24],
- window systems and user interfaces (Presentation Manager, X-Windows, Open Look), and
- hypertext and hypermedia [15]

make the use of different media in a computer environment feasible, now and in the future [18]. These components and media may be used and combined into a multimedia system for a wide range of applications, e.g., advertisement, tutoring, or support of traveling agencies.

This paper refers to "*multimedia*" as the integrated generation, representation, processing, storage, and dissemination of independent machine processable information expressed by means of multiple time dependent and time independent media. A medium denotes a type of information such as data, graphics, drawings, animation, voice, audio, and video. In audio, video, voice, and animation the information itself may be expressed as a function of time. It has to be pointed out, that these time dependent media take into account the characteristics of isochronous data streams (voice, audio, full-motion video). Text, graphics, and drawings are time independent media.

Consider an application which combines a series of images and speech fragments in order to explain a certain chemical phenomenon in different languages. The user has the additional possibility to speed up (skip the actual image and the respective audio information), slow down (look as long as he wants at the actual image), or repeat the explanation of the actual image. The interaction with the user is done via mouse or keyboard. Without the interaction with the user and the choice of different languages it is just like a slide-show with a tape running and ruling the projector (i.e., synchronizing the images and voice fragments). But the additional possible interactions requires more intelligence of the controlling device/s and fast random access to the stored information. This scenario relies on the independence of the media; e.g., it must be possible to play audio fragments without accessing images.

With this example the requirements regarding the *independence*, the *coexistence*, the *integration*, and the *interaction of different media* by mutual synchronization in a multimedia environment were outlined.

Another reason for introducing synchronization relates to the different communication paths for different media. Streams which convey data of different media may flow in a distributed environment on different channels or carriers (e.g., transatlantic cable or via satellite) with different propagation properties. Synchronization is required somewhere in the multimedia system nearby the sink of the multimedia streams.

"*Synchronization*" makes events happen in a certain time order. "Synchronization" in the context of multimedia refers to the mechanisms used by processes (also specific to multimedia) to coordinate their ordering in the time domain. The access to shared resources enforces exclusion from the resource and schedules access to the resource; i.e., synchronization solves the problems of specifying and controlling joint activity of cooperating

Manuscript received May 29, 1989; revised November 11, 1989. This work was carried out as part of a distributed multimedia system (DMS) project at the IBM European Networking Center in Heidelberg, Federal Republic of Germany. DMS will provide a system interface for multimedia resources in a heterogeneous and distributed environment. This paper was presented in part at the IEEE 5th International Workshop on Telematics, Denver, CO, September 18-21, 1989.

The author is with IBM European Networking Center, 6900 Heidelberg, Federal Republic of Germany.

IEEE Log Number 8934089.

processes, and of serialization of concurrent access to shared resources by many processes [17].

Synchronization and communication are often combined, synchronization differs from communication considering the mode of interaction.

"Synchronization" refers to the ordering of processes and their coordinated access to shared resources. It influences the behavior of the processes directly. This influence can be regarded as a special kind of information. The information transmitted is only the presence, absence, or ordering of processes. We do not claim this to be communication. Because "communication" involves passing information between processes, communication must rely on some kind of synchronization. Therefore, communication involves synchronization.

Communication will not be considered in this paper.

This paper covers distributed as well as central system configurations. Distribution is based on interconnected autonomous information processing nodes. The respective programs on these nodes may cooperate in order to achieve a common goal. Synchronization in a distributed multimedia system imposes further restrictions and problems to be solved. Nevertheless, all relevant issues of local and central systems are included. Therefore, we focus on the more general scenario, a distributed environment.

The following categories of media integration exist according to [1].

- *Intermedia Relationship:* A state transition or activity in one medium affects another medium, e.g., a certain text pattern activates a moving video sequence.
- *Media Conversion:* The information contained in one medium is "translated" into information in another medium. The text to speech conversion is an example.
- *Media Cooperation* means the simultaneous exchange of information of two or more media such as simultaneous audio and video transmission.

Synchronization is needed for all of the three categories, but to a very different extent. The intermedia relationship and media cooperation refer to the ordering of events and therefore to the basic synchronization. Media conversion involving synchronization as the relation between original and transformed patterns must be implemented using synchronization mechanisms.

The following example (see Fig. 1) shows a video stream generated by a camera, a voice stream generated by a tape recorder, and user interaction(s), which will be synchronized. The voice stream might consist of prerecorded voice fragments as general comments to an experiment of very short duration, e.g., a crash test with cars.

A certain voice fragment comments the technical aspects of crashes. These comments last much longer than the experiment itself. At the end of the video sequence the "multimedia object" related to the video stream wants to synchronize itself with the multimedia voice object. As the duration of the voice multimedia object is longer than the video sequence a "gap" occurs (see Fig. 2).

Usually no action is performed if processes or activities wait for an event to take place in order to synchronize

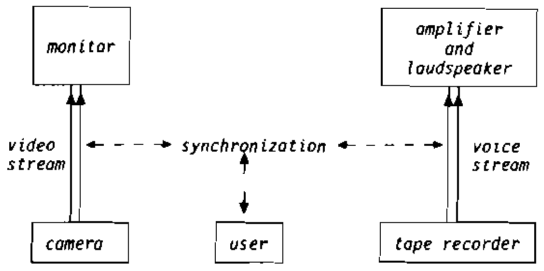

Fig. 1. Example showing the elements involved in synchronization in a multimedia system.

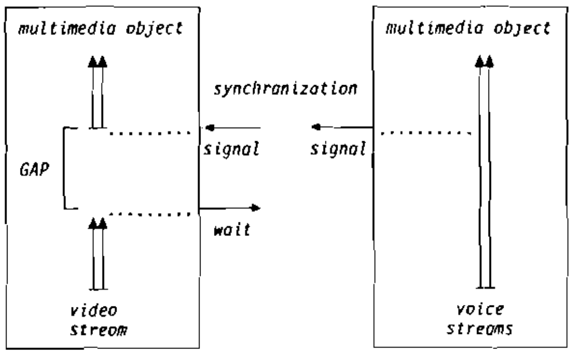

Fig. 2. Example of synchronization between two streams. The "gap problem."

themselves. Such processes and activities are usually suspended. But, what does it mean for multimedia streams to be "suspended"? What happens in the "gap"? The example of Fig. 2 shows a video stream which is to be synchronized with voice fragments. The video stream must "wait" for the voice stream. When the voice stream is ready for synchronization a "signal" is sent to the synchronization mechanism or directly to the video stream. Immediately, both streams continue to flow. But it has to be defined what will happen during the "gap" on the video monitor. The consideration of time dependent media in the context of synchronization requires the execution of actions, while a process or stream is waiting for a synchronization event. In our example, an immediate solution to the "gap problem" is to display the last picture of the video sequence. Certainly this is media, system, and device dependent. A more general solution called "*restricted blocking*," which also enables the specification and execution of other operations than the display of the last picture, is presented in this paper.

The most common way of presenting and discussing synchronization, usually combined with communication, is done by means of the existing concepts like: busy waiting, semaphore, monitor, message, remote procedure call. This paper follows a more fundamental approach by *classifying the essential characteristics* of synchronization. The classification is based on the synchronization characteristics in the context of operating systems and parallel programming languages. The scope is enhanced by multimedia specific requirements. The development of synchronization mechanisms for distributed multimedia systems must consider/incorporate these basic properties specific to multimedia outlined in this paper.

In this paper existing and new synchronization characteristics are presented. Special emphasis is placed on the incorporation of inherent multimedia features, rather than presenting new or enhanced interaction or synchronization mechanisms.

This paper covers synchronization in the context of operating systems and concurrent languages. It does not approach synchronization as understood by transaction in database environment (e.g., exclusive or share locks, and restart as concurrency control mechanism).

The following chapter defines an object-based model for a multimedia system, which is relevant for further discussions. In the next chapter, a classification of the characteristics of synchronization is presented. New requirements of multimedia environments are identified. The resulting enhancements to address the new requirements are described in detail in the following chapter. In the conclusion, the relevance of the achieved results is formulated showing the direction of further work to be carried out. The Appendix completes the classification scheme. An overview of the characteristics of the specification, creation and lifetime of objects is presented.

II. An Object-Based Model of a Multimedia System

This chapter presents and defines the elements for describing "synchronization." These will be used throughout the paper.

Usually an object is represented by the interface and not by internal details. Nevertheless, for didactical purposes in this paper the description and the related pictures of multimedia objects will be shown as composed of particular elements. The actual implementation may differ from this description. It is not the aim to imply a specific implementation at this level of discussion.

An object based model of a multimedia system—a *Multimedia Object System* is presented below (see Fig. 3). Only such components of a multimedia system which are essential for the discussion of synchronization are included in this model. These are as follows.

- *Activities:* Sequential unit of execution.
- *Multimedia Objects:* Sequential unit of execution combined with data specific to particular medium or media.
- *Interactions:* Relationship between activities and multimedia objects.

Activities and multimedia objects are the active components of the system. They are the *objects* of the system. An object is an element of information processing, whose state can only be retrieved and influenced by a well defined set of operations called *methods*. An object can be created and has a lifetime.

The interactions between objects involve synchronization. Objects execute *synchronizing operations* in order to assure a temporal order of events. The execution of cooperating synchronizing operations is called a *synchronizing event* (see Fig. 4); i.e., it indicates the action of synchronization of objects at a certain point in the list of consecutive actions of the multimedia activity or the activity, respectively.

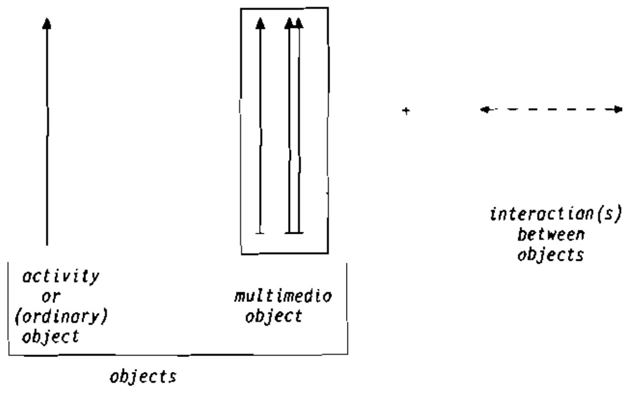

Fig. 3. Multimedia Object System.

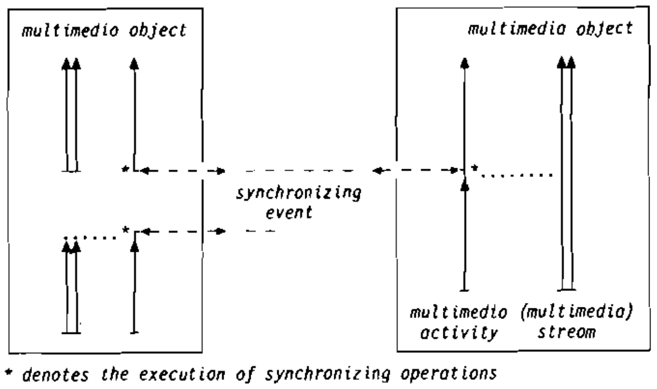

Fig. 4. Synchronization of multimedia objects.

Multimedia objects interact with other objects. Up to a certain extent they execute synchronizing operations as regular processes do. The required enhancements will be presented in Section IV.

A multimedia object consists of the following.

- The *Multimedia Stream:* The data specific for one or more media (e.g., sampled voice items or images as part of a full-motion video object) and
- The *Multimedia Activity:* The persistent information of the multimedia object and its behavior (e.g., characteristics of a surveillance camera with association to the allowed methods such as move or zoom). This is the activity of a multimedia object.

Multimedia objects could have the attributes outlined in the following paragraphs.

1) A multimedia object may contain synchronizing operations. This information about the synchronizing events comprises

- the involved partner(s), i.e., other object(s) or intermediate elements like a channel in CSP [14],
- the type of synchronization such as blocking or nonblocking,
- the determination of the release mechanism (e.g., certain picture of a series of images or a function of time).

2) Information about the synchronization operations can be retrieved and changed (according to the access rights) by other objects. It comprises the ability to

- retrieve all information about the own synchronizing events. Queries may address synchronizing events involving the object itself in the past, present, or even future:
- add synchronizing operations,
- change synchronizing operations,
- delete synchronizing operations, and
- suspend synchronizing operations for some time or for the object lifetime.

3) A central, global object called "synchronization manager" may impose synchronization on objects. The existence of such a controlling instance is a typical feature of local interactive tutoring systems. The synchronization manager activates the objects involved and imposes synchronization. The type of synchronization and the release mechanisms are sent as part of the information to the objects or are imposed on them from outside.

> A good example is a tutoring system which interacts with a student. It may involve pictures, moving video and voice. A control program defines the finite states and possible moves to new states, with activation and deactivation of the media.

Imposing synchronization (i.e., the existence of a global controlling instance) simplifies the programming of applications involving mainly local controlling features. An example is a simple tutoring system with one user. But it does not solve all synchronization problems in the most elegant and effective way, especially those between many distributed partners.

4) Multimedia objects may synchronize with other multimedia objects directly. No global instance is required. Direct synchronization may be the basis for the synchronization imposed by a controlling instance, as outlined in the preceding paragraph.

5) Multimedia objects may contain information about the medium itself, the medium type, and/or the device involved.

> Consider audio signals: The respective multimedia objects should provide a method to measure the volume of the signal. This would facilitate writing a real-time conference application. In this application the algorithm for resource allocation of a common audio output device relies on the speaker with the loudest recently spoken words.

6) Multimedia objects may exist in a local and in a distributed environment.

Finishing the description of the attributes of multimedia objects it has to be remarked, that this model is also suitable for analyzing and describing communication as interaction between the objects.

Up to now the description of the components of a multimedia object refers to a single multimedia activity and a single multimedia stream. This notion can be extended to a single multimedia activity and many multimedia streams. The relationship between different multimedia streams inside a multimedia object is fixed; i.e., further synchronization is neither necessary nor possible (see Fig. 5).

> A video tape containing voice and moving video signals does not need synchronization if the sound and the video are played physically close to the VCR. The respective multimedia object comprises one activity and two multimedia streams (voice and video). Consider a text mixed with a moving video and stored together on an optical videodisk. The respective multimedia object contains a single activity, the text, and the movie. The text and the video might not even be possible to separate.

III. Basic Synchronization Characteristics

Objects interact in order to reach one or many common goals. The interaction involves synchronization of the components by applying certain synchronization mechanisms. "Synchronization" can be characterized according to the criteria discussed in this section (part of them are discussed in [2], e.g.). The presented characteristics are derived from conventional synchronization and communication mechanisms such as semaphore, monitor, or RPC. To the knowledge of the author such a complete and fundamental taxonomy of basic characteristics has not been presented before. Some useful and necessary extensions especially for multimedia are identified and explained in this chapter.

The definition and elaboration of appropriate synchronization mechanisms require a selection between different properties of 8 basic characteristics. First, we want to outline some properties of the these characteristics by means of an example.

> Consider the synchronization mechanism of CSP with the concept of a rendezvous. It is expressed by input operations (denoted by a question mark after the name of the channel) and output operations (denoted by an exclamation mark after the name of the channel) on a channel.
> In the following example two parallel processes communicate via the channel 'comm':
>
> [. . .; comm ? x; . . . || . . .; comm ! a; . . .].
>
> The first process to reach the communication statement is blocked and waits for the partner. When the second process reaches the communication statement the value of 'a' is assigned to 'x' and both processes proceed.

The 8 characteristics of synchronization in CSP have the following properties.

1) There are always two processes involved as a point-to-point connection.

2) These processes know an intermediate object (the channel attached to both) as means of synchronization.

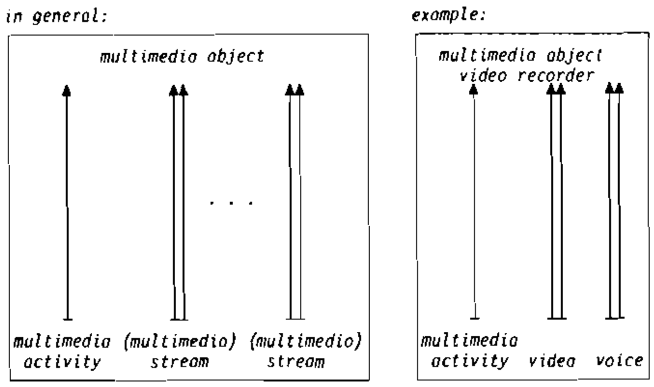

Fig. 5. Multimedia objects with more than one stream.

3) The first process waiting to communicate/synchronize is blocked.

4) A process waiting for a synchronization event cannot be preempted.

5) Many synchronizing events can be "combined" by an alternative guarded command.

6) In the alternative guarded command the choice between different possible synchronization events is predefined as nondeterministic.

7) The relation between both processes is symmetric.

8) No real-time aspects are considered.

The basic synchronization characteristics of object interactions are the number of involved objects, the naming, the behavior of individual objects while waiting for synchronization, the influence of those objects on others already synchronizing, if more than two objects are allowed to combine many synchronization events, to order synchronization events, the relationship between the interacting objects and the coverage of real-time aspects (Table I):

1) The first characterization item concerns the *number of involved objects*.

• The synchronization of two objects is fundamental to all synchronization concepts. This is a 1 to 1 relationship.

• The synchronization of more than two objects with respect to one other object covers the "one to many" and "many to one" relations.

If communication is involved in the interaction "one to many" and "many to one" should be differentiated: "many to one" identifies the relation between one server and many clients; whereas "one to many" (multicasting or broadcasting) is used by a single object to notify some condition(s) to other objects.

> Given a concurrent slide show (every slide is a monitor showing still images) all monitors are arranged as an two-dimensional array in front of the students. At a certain point a large picture will be shown. Every monitor displays part of the picture. The whole picture has to appear simultaneously with the creation of a multimedia object with a voice stream. In this situation the programmed relation between a single voice multimedia object and many image multimedia objects is "one to many."

TABLE I
SUMMARIZED BASIC SYNCHRONIZATION CHARACTERISTICS

characteristics	*attributes*			
number of involved objects	2 objects: 'one to one'	'one to many', 'many to one'	'many to many'	
behavior of individual objects waiting for synchronization events	blocking	non-blocking	***'Restricted Blocking' for multimedia objects required***	
naming of interacting object(s)	direct naming		Non-direct naming	
influence on other objects already synchronizing	influence		no influence	
combination of many synchronization events	no combination	combination at receiver	combination at sender	combination at the receiver and at the sender
order the executed synchronization operations of many synchronization events	predefined order	prioritized order	conditional order	prioritized and conditional order
relationship between interacting objects	symmetric		asymmetric	
real-time aspects	not considered		***required for multimedia objects***	

• The last case involves synchronization of more than two multimedia objects in a "many to many" fashion. But, up to now it is not obvious if this is relevant in the multimedia synchronization environment.

2) The *naming* of interacting object(s) refers to the manner of referencing the involved synchronization mechanisms or objects.

• In a direct naming convention the involved objects must be referenced directly by others. This implies the availability of knowledge about the involved objects (also in a client server model). Therefore, it is usually easier to verify the system behavior in a direct naming mechanism than in the indirect. Problems arise if the system configuration changes dynamically or objects are created at runtime.

• Indirect naming means naming of intermediary. partner element(s) such as a mailbox. An object does not refer to another object directly. The interacting objects do not need to know the object with which to synchronize. A semaphore belongs to this category.

3) The *behavior of individual objects waiting for synchronization events* covers blocking aspects. This is similar to the behavior of a transaction which attempts to lock objects already locked. It must either wait, abort itself or preempt the other transaction(s) involved.

• The *nonblocking* behavior would never force synchronization. But, it could allow the retrieval of information about cooperating objects such as "is another object in a specific state?".

> A message passing concept with a mailbox has a nonblocking characteristic.

• An object waiting for a synchronization event may be *blocked*:

> Consider the example of displaying an image of a fossil animal and add some explanatory text. The multimedia objects are the display of the image and the display of the text. Let the text object invoke a synchronization operation. For didactical purposes the image should be displayed before the text. Otherwise, the text would perhaps appear on top of another not convenient picture. The resource delivering the text is ready before the image may be retrieved from a central server. In this case, the object which generates the text (it might be the device driver or the multimedia object itself) must be blocked until the image object is ready to synchronize.

In the multimedia context it has to be defined what it meant by blocking a multimedia stream. A blocked full-motion video stream could mean to display the last image for the time of being blocked. If a voice stream is blocked, then no voice signal has to be played. Certainly, this definition depends on the hardware and system software. For displaying the last image of a full-motion video, a video image store is necessary (see Fig. 12). For a more general approach see "restricted blocking."

• *Restricted blocking* is a new behavioral concept required for some multimedia applications.

If processes wait for an event to take place in order to synchronize, then no action may be performed. Such processes may be suspended. What does this mean for a multimedia object? One alternative is that the multimedia stream should not be stopped. The concept of "restricted blocking" allows an exact definition of the behavior of the stream(s) while waiting for a synchronization event. This concept is explained in the following chapter dealing with the peculiarities for multimedia.

4) Individual objects to be synchronized may or may not have *influence on other objects synchronizing already*. This can occur if more than two objects are allowed to be involved in a synchronization event.

• The influence on other involved objects may lead to the preemption of the other objects. As a consequence the object itself never becomes blocked. This feature is common in a database environment and low level system programming. It can also be seen as nonmaskable interrupt.

• The usual behavior of synchronization mechanisms allows no influence of an object waiting to synchronize on those already synchronizing.

5) The involvement or *combination of many synchronization events* refers to a selective synchronization.

• There might be *no combination* of synchronization events defined at all. As a consequence for blocking mechanisms it is not possible to ask if the other object is ready for synchronization. In CSP without alternative guarded commands this would be the case. In a message-based communication environment this could mean the absence of "just look if a message is there, and if there is none then go on."

• If a *combination for receivers* (servers) may be defined, then receiving objects may wait on synchronization with multiple others. The select statement of ADA is an example.

• If a *combination for sender* (client) may be defined, then sending objects may wait on synchronization with multiple others. Remark: Pure synchronization does not distinguish between sender and receiver or client and server, i.e., this and the preceding paragraph describe the same feature.

• The *combination at the receiver* (server) *and at the sender* (client) serves as basis for symmetrical relationship between the objects. A typical example is the alternative guarded command ([8] and [13]).

6) The possibility to *order* the executed synchronization operations of many synchronization events is another classification item. If two or more synchronization operations are combined by some kind of relation the order may be specified in which such interactions should take place. This issue is similar to the ordering of process interaction requests ([12].

• A *predefined order* can not be altered by the programmer or user (e.g., it may be FIFO type or even nondeterministic).

> E.g.: CSP allows a nondeterministic selection between alternative guards. No ordering between the synchronized processes is imposed.
> Considering synchronization/communication in SDL [27] by signals a predefined FIFO queue organization at the receiver is specified.

• *Prioritized order* means the influence of synchronization by attaching priorities to different synchronization events or objects.

> In the following example of Concurrent C [11] the expression "prio" is evaluated for all outstanding synchronization calls and the call with the minimal value is accepted:

```
. . .
accept name_of_outstanding_transaction (param1,
  param2, . . . ) by (prio)
. . .
```

• A *conditioned order* is more flexible than the prioritized:

> Concurrent C allows a condition "cond" involving the parameters "param1," "param2," . . . to be evaluated (see the following example). The first outstanding call where the condition 'cond' is true is accepted (i.e., the synchronization event occurs).

```
. . .
accept name_of_outstanding_transaction (param1,
  param2, . . . ) suchthat (cond).
. . .
```

• *Prioritized and conditional order* may be combined.

7) The *relationship between interacting objects* can be *symmetric* or *asymmetric*. Different symmetric or asymmetric relationships between calling and called parties can be distinguished.

• The most frequent known aspect relates to the naming of objects.

> Consider the rendezvous in ADA: The calling "client" task must name the called "server" task. The server task identifies the entry points by accept statements without identifying the calling task.
> In contrast processes synchronizing on the basis of the rendezvous of CSP explicitly name each other and no differences exist between calling and called process/object naming.

Another aspect of asymmetry is, e.g., the ability to serve as a guard (see [10]).

> Consider ADA again: The server task has the ability to specify a guard by the select statement. There is no such feature supported by the calling task.
> The input and output operation of CSP both allow guards.

• An asymmetric relationship exists if the state of interacting activities or objects is blocked, nonblocking or restricted blocked depending on calling or being called.

> An example for asymmetric relation in this context is SDL. In SDL the process instance sending a signal is never blocked, the process expecting a signal may be blocked.
> Synchronizing by semaphores is completely symmetric: calling and called activities may be blocked.

8) Synchronization may or may not involve *real-time aspects* as part of the mechanisms.

• Most of the known principles of synchronization and communication have *no real-time aspects*.

• As multimedia deals with data streams conveying information to be passed in real-time (e.g., voice) the synchronization of such elements incorporates *real-time* features. See next chapter for details.

Synchronization can also cover other aspects not to be considered in this paper such as the synchronization of video images at composite video signals level. These features are assumed to be done by the hardware of the respective devices and are not relevant for synchronization in our context. All discussed items are summarized in Table I.

IV. Enhancements for Multimedia

The discussion of the basic properties of synchronization mechanisms revealed two items necessary in the multimedia context. "Restricted blocking" and real-time aspects. Synchronization in multimedia systems can be discussed at the level where the synchronization entity is a bit or byte, rather than a whole picture or text paragraph. Then "continuous synchronization" may be introduced.

A. Restricted Blocking

Restricted blocking is a new behavioral concept required for some multimedia applications.

If processes wait for an event to take place in order to synchronize no action may be performed. Such processes are usually suspended. What does this mean for a multimedia object?

The first multimedia object executing a synchronization operation waits for the synchronization event because the synchronizing condition fails. In this sense the object is blocked. While waiting the multimedia object is not suspended, it performs an alternative action. This action is aborted as soon as the synchronizing condition rises.

The multimedia object is composed of the multimedia activity and the multimedia stream. The respective multimedia activity may be suspended. If the multimedia object is blocked, then the multimedia stream may be stopped. There may be no video signal and no audio signal. But sometimes there should be a multimedia stream to fill the gap (see Fig. 2).

In the video context the stream to fill the gap may be the last picture or a (moving) video. The video signal can either be derived from the same source as where the stream originated before the execution of the synchronization operation, or it can be derived from an alternative source.

Without restricted blocking the solution of blocking multimedia streams for short-time duration is device and media dependent. A short-time duration refers typically to about a few milliseconds. Transient objects contain a stream, which are not retrieved from a store. The source of such a stream may be a camera or a microphone. As shown in Table II such a stream would continue to flow even if the respective activity of the multimedia object is blocked.

A persistent object or details of such an object can be retrieved from a storage. Therefore, it becomes possible to display the last picture (see Table III) again. A transient object can be stored in a buffer and becomes then, up to a certain extent a persistent object. A long-term gap lasts typically longer than a few seconds. Such a gap is usually filled explicitly by a stream, i.e., the programmer implemented a dedicated solution. This is also included in "restricted blocking."

A medium and device dependent amount of buffer storage is required to allow a flexible implementation of the restricted blocking concept. The largest amount of memory is needed for full-motion video. If a video stream has to be synchronized due to different delays of different medium transmission channels some images might be stored. This synchronization issue may be part of the transport protocol [23]. In the case of full-motion video always the last image should be stored. According to the CCIR-601 coding standard about 1 Mbyte of storage is required:

$$(13.5\text{ MHz} + 6.75\text{ MHz} + 6.75\text{ MHz})$$
$$* \; 8\text{ bit}/8\text{ bit/byte}/25\text{ images/s} = 1.08\text{ Mbyte}.$$

TABLE II
EXAMPLES OF TRANSIENT STREAMS

medium	*device*	*behavior of individual objects waiting for synchronization events*
acustic: audio	microphone	non-blocking
visual: full-motion video	camera	non-blocking

TABLE III
EXAMPLES OF PERSISTENT STREAMS

medium	*device*	*behavior of individual objects waiting for synchronization events*
acustic: audio	CD-ROM	blocking: no signal delivered from source
visual: full-motion video	VCR	blocking: no signal delivered from source or last image displayed

The following program shows a synchronization between an object showing full-motion video and audio signal about a physical experiment. If object *A* (full-motion video) reaches first the synchronization point the last image shown will be displayed. Alternatively if the audio object executes the synchronization operation first then some music will be played.

```
program of object A:          program of object B:
-- full-motion video          -- audio
--   from WORM                --   from CD-ROM
display (experiment_fluid)    play (experiment_fluid)

. . .                         . . .
SYNCHRONIZE                   SYNCHRONIZE
  WITH object_B                 WITH object_A
  AT end                        AT end
  MODE                          MODE
    restricted_blocking           restricted_blocking
  WHILE_WAITING                 WHILE_WAITING
    display_last_image            play (music_Bach)
  . . .                         . . .
```

The Fig. 6 sketches the alternative where the movie showing the water is displayed while waiting for the audio signal.

If two or more media are combined in a multimedia object, the behavior for each of them individually or together has to be defined in the "WHILE_WAITING" section. It has to be pointed out that this is not busy waiting. The object terminates the alternative WHILE-_WAITING section if the synchronization rises. This operation is initiated by the system and not by the object itself as it would happen in a busy waiting fashion.

B. Real-Time Basic Synchronization Semantics

The synchronization of multimedia objects may be "optimal," "good," "acceptable," or even perhaps "not tolerable." This depends on the characteristics of the streams involved, the actual application or system program, and the time relation between the involved executed synchronization operations. There has been no need to incorporate this feature into the semantics of "process synchronization" in the operating systems environment if multimedia is not part of the system.

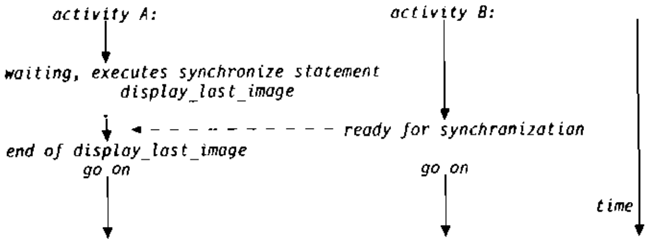
activity A:
activity B:
waiting, executes synchronize statement
display_last_image
ready for synchronization
end of display_last_image
go on
go on
time

Fig. 6. Synchronization with restricted blocking.

The *event stamp* (see Fig. 7) is defined as a point of synchronization in the execution of a synchronization operation by an object. It must occur within the lifetime of an object.

A synchronization event relates to at least two event stamps of different objects. In the general case the event stamps may be anywhere in the lifetime of the objects (see Fig. 8).

Within the lifetime of the object there are two special locations of an event stamp.

- The event stamp is the creation point of the object.
- The event stamp is the end of the lifetime of the object.

If the event stamp on both objects are placed at the beginning, the synchronization is called "parallel" or "simultaneous" (see Fig. 9) [6], [20].

If the event stamp of one object occurs at the beginning and the other at the end the synchronization is called "serial" or "sequential" (see Fig. 10), it can be viewed as a kind of concatenation [6], [20].

The synchronization and communication mechanisms used in the operating system environment usually do not support time functions. Therefore, nothing is said about the "delay" between two event stamps to be synchronized. The situation differs completely in the multimedia context, e.g., it matters if the comment related to an event shown as moving pictures is delayed too much. Experiments at CCETT (France) have shown, that some applications require picture and sound to be synchronized within around 150 ms [6].

Concerning the delay the following terms of multimedia object relations are introduced.

- *Timemin* is the minimal acceptable delay between two synchronizing event stamps. The usual parameter is "0."

E.g., timemin (event-stamp-*A*, event-stamp-*B*) = *n* ms is read as the minimal delay of event-stamp-*A* with respect to event-stamp-*B* is *n* ms.

- *Timeave* is the ideal delay between synchronizing event stamps; i.e., the second synchronization event stamp should be as close as possible to timeave, it might happen before or after timeave. Usually timeave is also 0.

E.g., timeave (event-stamp-*A*, event-stamp-*B*) = *n* ms

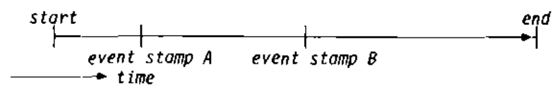

Fig. 7. Event stamp.

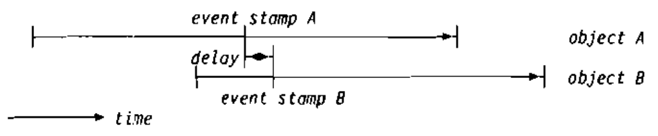

Fig. 8. Relation between event stamps—general case.

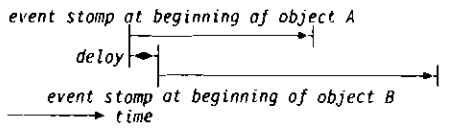
event stamp at beginning of object A
delay
event stamp at beginning of object B
time

Fig. 9. Relation between event stamps—"parallel synchronization."

is read as the average delay of event-stamp-*A* with respect to event-stamp-*B* is *n* ms.

• *Timemax* or "time-out" refers to the maximal acceptable delay between synchronizing event stamps. Usual parameters vary depending on the media involved and on the task to be performed.

E.g., timemax (event-stamp-*A*, event-stamp-*B*) = *n* ms is read as the maximal delay of event-stamp-*A* with respect to event-stamp-*B* is *n* ms.

The operands "timemin" and "timemax" are not commutative. This characteristic provides the power to specify a different delay depending on which multimedia object executes its synchronizing operation first. The functions timemin, timeave, and timemax can be combined by the logical AND operator (in special cases some of the expressions are equivalent).

expression	*meaning*
timemin (. . .)	see above.
timemax (. . .)	see above.
timeave (. . .)	see above.
timemin (. . .) AND timemax (. . .)	synchronization should take place between timemin and timemax (time interval).
timemin (. . .) AND timeave (. . .)	synchronization should take place as close as possible to timeave, but never before timemin.
timemax (. . .) AND timeave (. . .)	synchronization should take place as close as possible to timeave, but never after timemax.
timemin (. . .) AND timemax (. . .) AND timeave (. . .)	synchronization should take place as close as possible to timeave, but between timemin and timemax.

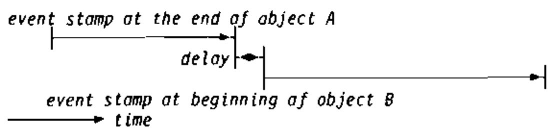
event stamp at the end of object A
delay
event stamp at beginning of object B
time

Fig. 10. Relation between event stamps—"sequential synchronization."

After introducing timemax, the exception of a delay greater than timemax has to be handled and incorporated into the concept. The same applies to timemin. The exception defines a set of actions. This exception handling is similar to the definition of the behavior of multimedia streams in the restricted blocking concept.

The following example is based on the program of Fig. 6.

```
program of object A:            program of object B:
-- full-motion video            -- audio
--   from WORM                  --   from CD-ROM

display (experiment_fluid)      play (experiment_fluid)

 . . .                           . . .
SYNCHRONIZE                     SYNCHRONIZE
  WITH object_B                   WITH object_A
  AT end                          AT end
  MODE                            MODE
    restricted_blocking             restricted_blocking
  WHILE_WAITING                   WHILE_WAITING
    display_last_image              play (music_Bach)

  TIMEMIN θ                       TIMEMIN θ
  TIMEMAX 1 s                     TIMEMAX 2 s
  TIMEAVE θ
  EXCEPTION                       EXCEPTION
    display_last_image              play (music_Bach)
 . . .                           . . .
```

It shows a description where the synchronization delay between the two objects should be between 1 s of the audio object ahead of the video object and 2 s of the video object ahead of the audio. This range is sketched in Fig. 11. The target delay between both should be 0, i.e., no delay. The synchronization mode characterizes the behavior in the allowed range. The exception denotes what happens out of this range. Often the functionality remains, as outlined in this example.

An alternative description of synchronization dependencies can be done by using the three operators sequential, simultaneous, and independent. In [20], these operators serve to order multimedia mailing documents. In [21], the elements of a multimedia document presentation are concatenated by these "temporal operations." The semantics presented in this paper cover all expressive power of the above mentioned operators. Indeed, with these three operators there is only synchronization at the beginning and end of multimedia objects (sequential, simultaneous). The "independence" is specified by the structuring of the objects (see the Appendix) as shown in the following example, where multimedia_object_text and multime-

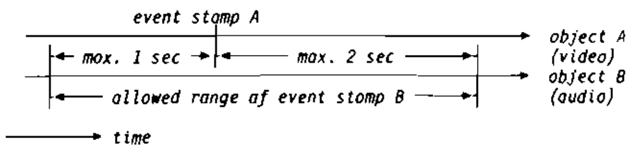

Fig. 11. Synchronization with real-time constrains.

dia_object_voice are independent:

```
COBEGIN
    multimedia_object_text
    multimedia_object_voice
COEND.
```

C. Continuous Synchronization

Synchronization of a multimedia object can be achieved by delaying the stream. The stream will not be blocked for the entire duration of the synchronization event, but each unit of information will be blocked/delayed for a certain period of time. This method requires a storage to act as buffer. In a video stream environment this would be possible with an intermediate video image store between the camera and the monitor as shown in Fig. 12.

The capacity of this intermediate storage with respect to the multimedia stream involved limits the use of this *continuous synchronization*. This synchronization could be implemented by the incorporation of synchronization markers within the data and an additional synchronization channel as described in [23].

The fundamental difference to the restricted blocking concept is the level of synchronization as shown in the following Table IV. The unit of information refers to the granularity of the multimedia stream. Synchronization can be performed between pixels of a video stream or, e.g., between whole full-screen pictures. "Continuous synchronization" deals with the lowest levels of granularity (e.g., bytes, pixels, speech samples) whereas "restricted blocking" is concerned with more complex entities such as full-screen pictures or paragraphs of text. Each row of the Table IV covers the entire range of granularity. But only examples of media and units of information are outlined in the table. The columns refer to certain ordering of the units of the individual rows. No relation/ordering exists between adjacent fields of different rows such as "bit" of "data/basic information structure" and "sample/pixel" of "full-motion video."

In this chapter the enhancements to the basic characteristics of synchronization for multimedia were presented.

V. Conclusion

This paper presented a fundamental survey of basic characteristics of synchronization mechanisms. It was shown that a distributed multimedia system requires further characteristics. Therefore we developed the concept of "restricted blocking" and incorporated real-time semantics in synchronization of multimedia objects. For the description of the above mentioned problems and solutions a multimedia object model was introduced.

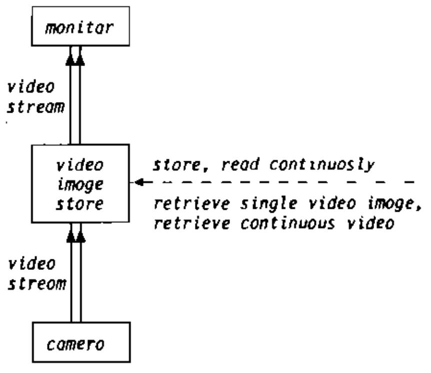

Fig. 12. Intermediate video storage.

TABLE IV
Media Dependent Unit of Information

media	*unit of information (level)*			
	suitable for 'continuous synchronization'		*to be distiguished for the discussion of 're-stricted blocking'*	
data / basic information structure	bit	byte	packet	whole transmitted information
full-motion video	sample / pixel	line or cluster	one full-screen picture	whole video information
coded full-motion video	sample	coded cluster	one full-screen picture	whole video information
image	pixel			whole image
drawing			individual objects	whole drawing / object tree
text (content)	character	word / sentence	paragraph	text document
spreadsheet	cell with formula			whole spreadshee
audio: speech	sample	allophon / phonem	word / sentence / paragraph	whole audio information
audio: music	sample		note	whole opus

On the basis of the properties outlined in this paper (i.e., taking into account restricted blocking and real-time semantics) existing synchronization mechanisms can be altered or new ones defined for multimedia environments. These mechanisms can be incorporated into a specification method or directly into an implementation language by a change of the syntax and/or semantics, by appropriate subroutines, or by the respective library modules. The constructor of such mechanisms should keep in mind requirements (see [5]) concerning the modularity of multimedia objects, expressive power of the concept itself and, ease of use.

As last example look at the correlation between music sound and written notes in a multimedia environment. According to the music faculty at Brown University (US) (see [25]) the use of multimedia would revolutionize the teaching of music. It would be possible to connect written notes to interpreted musical sounds in an integrated music editor, a piano keyboard, and a synthesizer. Apart from conversion mechanisms (written notes to music sound and perhpas vice versa) the basic feature in this context will be an efficient synchronization mechanism which incorporates real-time aspects as those described in this paper.

Appendix
The Specification, the Creation, and the Lifetime of Objects

In order to complete the taxonomy of synchronization characteristics in this chapter we describe the properties of specification, creation and lifetime of objects.

Objects can interact only if they act at least partially in parallel. The environment must provide the ability to create and finish (delete, kill) objects. There is no need for any special feature to be provided in the multimedia context, but in this chapter we want to briefly survey the possibilities; i.e., the described multimedia object model is not restricted to any of the following categories.

1) The process of *creation* of interacting objects can be initiated in many different ways. The creation covers concepts like coroutines as well as objects started on different processors. Remark: Coroutines transfer the control by "call called-process" and "resume calling-process" (see, e.g., Simula [7]) and do not support real parallelism.

• The creation of objects may occur *implicitly at program start, at module entry* time or as a result of the *execution of statements.* As typical construct is "cobegin . . . coend." All statements or processes (at the first level of hierarchy) inside this construct are executed in parallel. An additional feature of "cobegin . . . coend" is the "single entry and single exit" property. All processes defined inside can only be created by execution of exactly this cobegin statement. The whole statement finishes after all internal processes have terminated. In CSP, this process creation method is used. The programmer is immediately confronted with many parallel processes.

> ADA and SDL do have such build-in mechanisms; e.g., in ADA a task type can be specified and then task variables can be declared. After the "block entry" these tasks are created (i.e., become active).

• The alternative is the *explicit creation* by some kind of create/start statement. The explicit creation enables sophisticated programming with the possibility of specifying complex relationships.

> Programming languages like ADA or CHILL and specification methods like SDL provide such a process creation ability.

2) The *power of creation* considers the point/time of object creation, i.e., "the degree of knowledge about objects at compile time." Do we know at compile time how many objects will exist at run time? Of course, this may depend upon certain parameters, but these parameters may be fixed at compile time.

• In the *static* case we have the knowledge at compile-time. The original version of CSP allowed only static process creation. Multimedia deals with physical resources (e.g., a microphone); multimedia objects are attached to such resources. The knowledge of real resource allocation requirements at compile time enables a more sophisticated program test and validation.

• Most of the specification methods and parallel programming languages allow *dynamic* object creation. The amount of objects is determined at run time. A more flexible handling of the multimedia objects and real resources is possible. SDL and ADA allow dynamic process creation.

3) The *lifetime* of the interacting objects stops (terminates) according to very different rules. An object may terminate

• by reaching the end of the code (module, program),

• by the processing of a terminate statement allowed to be executed by the object itself,

• by the processing of a terminate statement allowed to be executed by the parent object (e.g., kill ..object-id..),

• by the processing of a terminate statement allowed to be executed by any object with appropriate rights, and

• only if all all related objects (e.g., sons) have finished (e.g., "cobegin .. coend").

As analyzed in the context of our project no extension specific to multimedia is required concerning the creation and the lifetime of objects.

Acknowledgment

The author gratefully acknowledges the critical comments made by K. Geihs, H. Schmutz, U. Hollberg, B. Schöner, and H. Stüttgen and the various discussions held with D. Shepherd and M. Wasmund, which contributed to the quality of this paper. Thanks to the advices of the anonymous reviewers, J. Dinger and J. Rückert the paper improved significantly in the final version.

References

[1] E. J. Addeo, A. B. Dayao, A. D. Gelman, and V. F. Massa, "An experimental multimedia bridging system," in *Conf. Office Inform. Syst.*, Mar. 23-25, 1988, Palo Alto, CA, SIGOIS Bulletin, vol. 9, no. 2 and 3, pp. 236-242, Apr. and July 1988.
[2] G. B. Andrews and F. B. Schneider, "Concepts and notations for concurrent programming," *ACM Comput. Surv.*, vol. 15, pp. 3-43, Mar. 1983.
[3] *The Programming Language ADA, Reference Manual*, American National Standards Institute, Inc., ANSI/MIL-STD-1815-1983, Lect. Notes in Comput. Sci., no. 155, Springer-Verlag, 1983.
[4] F. Bican, "The worm turns," *PC Mag.*, pp. 199-224, Mar. 1988.
[5] T. Bloom, "Evaluating synchronization mechanisms," in *Proc. 7th Symp. Oper. Syst. Principles*, Dec. 10-12, 1979, Pacific Grove, CA, pp. 24-32.
[6] CCETT, "Multimedia synchronization," CCETT Int. note: AFNOR adhoc group on AVI standardization, July 1988.
[7] O. J. Dahl, B. Myrhaug, and K. Nygaard, "Simula 67- Common base language," Norwegian Comput. Cent., Rep. 743, 1984.
[8] E. W. Dijkstra, "Cooperating sequential processes," in *Programming Languages.* F. Genuys, Ed. New York: Academic, 1968.
[9] S. R. Faulk and D. L. Parnas, "On synchronization in hard-real-time systems," *Commun. ACM*, vol. 31, no. 3, pp. 274-287, Mar. 1988.
[10] N. Francez and S. A. Yemini, "Symmetric intertask communication," *ACM Trans. Programming Languages Syst.*, vol. 7, no. 4, pp. 622-636, Oct. 1985.
[11] N. H Gehani and W. D. Roome, "Concurrent C," Software—Practice and Experience, vol. 16, no. 9, pp. 821-844, Sept. 1988.

[12] ——, "Rendezvous facilities: Concurrent C and the ADA language," *IEEE Trans. Software Eng.*, vol. 14, pp. 1546-1553, Nov. 1988.
[13] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, pp. 666-677, Aug. 1978.
[14] ——, *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentiee-Hall, 1985.
[15] Special Issue on Hypertext, *Commun. ACM*, vol. 31, no. 7, July 1988.
[16] S. van Kervel, "Laservision ROM systems for digital data and video storage," *Comput. Standard Interfaces*, vol. 8, pp. 83-88, 1988.
[17] W. H. Kohler, "A survey of techniques for synchronization and recovery in decentralized computer systems," *ACM Comput. Surv.*, vol. 13, no. 2, pp. 149-183, June 1981.
[18] B. W. Mel, S. Omohundro, A. Robison, S. Skiena, K. Thearling, L. Young, and S. Wolfram, "Tablet: Personal computer in the year 2000," *Commun. ACM*, vol. 31, no. 6, pp. 639-646, June 1988.
[19] R. M. Newman, Z. L. Budrikis, and J. L. Hullet, "The DQDB MAN," *IEEE Commun. Mag.*, vol. 26, no. 4, pp. 20-28, Apr. 1988.
[20] J. B. Postel, G. G. Finn, A. R. Katz, and J. K. Reynolds, "An experimental multimedia mail system," *ACM Trans. Office Inform. Syst.*, vol. 6, no. 1, pp. 63-81, Jan. 1988.
[21] A. Poggio, J. Garcia Luna Aeeves, E. J. Craighill, D. Moran, L. Aguilar, D. Worthington, and J. Hight, "CCWS: A computer-based, multimedia information system," *IEEE Comput.*, pp. 92-103, Oct. 1985.
[22] F. E. Ross, An overview of FDDI: The fiber distributed data interface, *IEEE J. Select. Areas Commun.*, vol. 7, no. 7, pp. 1043-1051, Sept. 1989.
[23] M. Salmony and D. Shepherd, "Extending OSI to support synchronization required by multimedia applications," *IBM ENC Tech. Rep.* no. 43.8904, Apr. 1989.
[24] Special Issue on the TRON Project, *IEEE Micro*, vol. 7, no. 2, Apr. 1988.
[25] N. Yankelovich, N. Meyrowitz, and A. van Dam, "Reading and writing an electronic book," *IEEE Comput.*, vol. 18, no. 10, pp. 15-30, Oct. 1985.
[26] P. Vary and R. Hofmann, "Speech codec for the European mobile radio system," *Frequenz*, vol. 42, no. 2-3, pp. 85-93, Feb.-Mar. 1988.
[27] CCITT "The CCITT specification and definition language," CCITT recommendation Z.100, 1988.

Ralf Steinmetz (S'83-M'86) was born in 1956 in Santiago de Chile. He studied electrical engineering at the University of Salford, England, and at the Technical University of Darmstadt, West-Germany, where he received the M.Sc. (Dipl.-Ing.) degree in 1982. Till 1983 he received a scholarship sponsored by the German PTT (DBP). Working as scientific assistant where he received the Ph.D. degree (Dr.-Ing.) in 1986 at the same university.

At the University of Darmstadt he worked in the area of Petri-Nets and concurrent languages. After the last graduation (Ph.D.) he investigated CSP and OCCAM and published the first book on OCCAM-2 in German. Then he joined the Advanced Development Department of Philips Kommunikations Industrie in Siegen-Eiserfeld, West Germany, where he was involved in an ISDN project and an ESPRIT projeet eoncerning a multimedia workstation. Since 1988 he has been Research Staff Member of the IBM European Networking Center in Heidelberg, West Germany. There he is involved in a multimedia project in the distributed systems research department. He lectures at the University of Frankfurt. His current research interests include formal specification and analysis, multimedia systems, distributed systems and operating system aspects in those environments.

Dr. Steinmetz is member of IEEE Computer and Communieations Society, ACM, GI, and ITG.