IEEE ICC'90, SuperCom'90
Atlanta, April 1990

[SSSW90] *Ralf Steinmetz, Hermann Schmutz, Bernd Schöner, Michael Wasmund;* **Generic Support for Distributed Multimedia Applications;** IEEE ICC'90, SuperCom'90, Atlanta, April 1990.

Generic Support for Distributed Multimedia Applications

Ralf Steinmetz, Hermann Schmutz, Bernd Schöner, Michael Wasmund

IBM European Networking Center,
Tiergartenstr. 8, 6900 Heidelberg, P.O.Box 10 30 68,
Federal Republic of Germany

Abstract

This paper addresses the question of how to build a convenient application programming interface for multimedia applications in a distributed and heterogeneous environment. According to the objectives of distribution transparency with autonomy of nodes, presentation transparency, device independence and the handling of resource allocation by the underlying system, a concept is developed for this new area of applications.

The model comprises resources operating as sources and sinks of transient and persistent information. These resources and their interactions appear as capabilities at the application programming interface. Applications do not distinguish between local and remote operations and resources. Authorization is integrated as the protection attribute of the capabilities. The model supports a level of abstraction where unnecessary details - such as device dependencies - are made transparent to application programs.

1. Introduction

The role of a computer is changing from a "computational" machine towards an "information processing" machine which supports the processing, communication and presentation of information. Information processing with "text" was complemented by graphics and images and is being enhanced to cover spreadsheets, formulas, audio, and video. New storage technologies and architectures allow to store different media in a digital computer-controllable manner, making a wide variety of multimedia applications feasible. Local information processing systems have been already available for several years running applications like hypercard [Yank85, Kahn88]. The scientific community has also presented several multimedia systems and applications with attractive functions ([Ludw87, Ghaf88], Athena project at MIT). However, most of them are dedicated systems and do not make use of high-speed interconnections to other systems. Problems of heterogeneity, authorization and associated protection schemes have not been considered.

With the advent of high speed networks such as DQDB, FDDI, B-ISDN or IBCN distributed applications over local, metropolitan or wide area networks will open the door for a rich set of novel applications. Examples are joint editing, tutoring with audio, video and data communications, travel and transport booking with the opportunity to view a video clip of potential destinations. or remote error diagnosis of a manufacturing process. Cooperative work in real time between journalists who jointly edit a newspaper article with the respective pictures, chemists designing a chemical product or engineers involved in CAD are other examples. The cooperating partners may be physically distributed.

Applications are executed in computers with attached storages for audio, video and data, sensors (camera, microphone), reproducers (loudspeaker, display), and associated control functions (e.g., positioning of sensors, gain control, adjusting the illumination). The computers may be workstations as well as mainframes and thus may be differently equipped. Workstations may have cameras attached to it, mainframes are expected to have banks of storage devices. Furthermore, the attached devices may have different data encoding, i.e. different conventions of analog and digital encoding. Special equipment to perform necessary conversions exist in the network, as well as equipment for mixing media of single or different types. Computers are linked by a network capable of transmitting audio and video streams in isochronous digital form; in addition, data communication allows for transfer of stored objects (still images, text, etc.) and remote control of devices.

It is widely believed that distributed multimedia applications are useful. Before a qualified statement on the usefulness of distributed multimedia applications can be given, we see three open questions:

1. To what extent are multimedia applications found desirable by potential users ?
   People will only be willing to pay the price for the attachment of additional devices if the expected benefits pay off in their view. By experience, we know it is very hard to predict what is ultimately liked or disliked by users. On the other hand, attachments will only be cheap if found attractive by a large community of users.
   Given the user's interest, there is immediately a second question:
2. Can the apparent complexity of multimedia systems be mastered such that user-friendly applications can be designed, implemented, tested and maintained at a reasonable cost ?
   It is known that distributed data processing applications are already complex. The multitude of devices

and control operations in multimedia systems adds a further dimension to this complexity.
Assuming that the complexity can be mastered, a third open question has to be answered.

3. What are the right primitives to be offered at the application program interface (API) ?

We don't claim to have a complete answer to any of the questions raised above. Instead, we suggest to find these answers by systematic experiments and analysis in realistic scenarios based on prototypes driven by ultimate end-users. We advocate to build early prototypes to gain as much experience and feedback as possible, to have the right answers ready when the expected high bandwidth networks become available. The approach presented in this paper is a first step towards answering the above questions.

We will present a concept for the development of an API for distributed multimedia applications. The proposed approach is driven by some fundamental design objectives like distribution transparency with node autonomy, presentation transparency, device independence, real resource allocation, and incorporation of synchronization mechanisms.

A different kind of API is considered in [Leun88, Leun89]. There, a set of operating system primitives based on the notions of active devices and connectors are presented to deal with multimedia devices in a UNIX environment. The functionality of UNIX primitives is enhanced with the goal of getting a UNIX style API. In contrast, our objectives were independence of a specific operating system and use in a heterogeneous multivendor environment.

In the following section we describe the design objectives. The model for multimedia processing is presented in section 3. We explain the concept and illustrate its usefulness by means of an example. In the the conclusion we summarize the results and indicate the future direction.

2. Design Objectives

The support for distributed multimedia applications presented in this paper was derived with a well defined set of objectives in mind. We combine well accepted objectives from the distributed systems area with additional objectives derived from the multimedia environment.

Distribution Transparency

In a local multimedia system the operations on the multitude of possible different attachments, like the control of a camera and its interconnection with monitors, may already be a complicated task, though the device names are known by a proper configuration process, and the links between the devices may be initiated by calls to device drivers.

In a distributed environment a video signal derived from a camera may be displayed at different remote locations. This is more complex due to the involvement of several distributed instances like local and remote processes and devices. Actions concerning the network with the respective communication protocols are involved. The name of a camera must be made known to the applications located on other workstations. The video stream originated from the camera has to be routed from one workstation via a high speed network using proper protocols to the other stations which in turn have to be set up properly to receive the video information.

From the application programming point of view it is desirable to hide these details by providing support from the underlying system. The access to remote resources should be as easy as the access to local resources, in the ideal case there should be no difference; i.e. the distribution should be transparent. However, it is not desirable to make all resources within the network accessible to everyone, because the local user wants to maintain full control over the resources in his environment. Consider, for example, the microphone of a multimedia workstation located in an office. The respective office worker would not like the idea that others can listen at any time. Therefore, the distributed system must provide mechanisms to control access to local and remote devices. The local node has to remain autonomous and has to maintain control over its local resources. However, it should be easy to give access rights temporarily to remote nodes.

Presentation Transparency

A distributed system may consist of heterogeneous hardware and software components; i.e. the attached devices and the operating systems as well as the involved machine architectures may be different. Prototypes of interconnected heterogeneous systems without any multimedia devices already exist and their results may be extended for multimedia systems [Krue88, Notk88]. But additional effort is required concerning the attached devices.

In the envisaged environment, media streams from various sources may be directed to multiple types of sinks. Consider a scenario with a NTSC camera which should be connected to a monitor working at the PAL standard. In this case a NTSC / PAL data stream converter is necessary. This expensive converter might be located and administrated by a central service at a different node. Presentation transparency requires that the involved applications should not have to handle this conversion and should not even need to know about the conversion and the location of the converter. All kind of necessary adaptations should be initiated and supervised by the distributed multimedia system software. The goal of presentation transparency is to shield the application from the underlying device heterogeneity. The system is expected to perform the required conversions.

Device Independence

The variety and large amount of attached devices implies a huge set of functions which may be supported by these devices. Similar devices with the same fuctionality may be operated with different interfaces at the hardware level. The application program should be independent of the actual system configuration up to a maximum extent.

It would be unacceptable to reduce the support of operations applicable to certain device types to a common subset. For example, all loudspeakers in the network may have the function 'set volume', but only a few also pro-

vide the function 'set tone'. Or, all cameras have focus setting, only a few of them allow zooming. These 'extra' functions would be excluded which is clearly not desirable. Device independence, as understood here, requires that application programs should be able to take advantage of available functions, but can also cope with situations where the full set of functions is not available.

Real Resource Allocation

In conventional data processing systems, we are accustomed to handle resources by provision of virtual resources which are multiplexed on real resources. For example, writing spool files may be viewed as virtual printing. In general, it is not necessary to have exclusive or non-preemptive use of a real resource.

Many multimedia applications require devices to be allocated as real resources. It does not make sense to share a microphone by a time slicing mechanism. Devices are often allocated for a longer time period. Mechanism to guarantee controlled and deadlock-free access to these devices are required. Either adequate reservation strategies have to be developed or deadlock detection and recovery has to be supported by the system.

Synchronization of Multimedia Objects

Isochronous data streams are generated by input devices like cameras which have no or very restricted storage properties and are fed into output or intermediate devices. Such data streams may have to be synchronized with other data streams or ordinary processes. This is a new requirement particularly for multimedia systems. For example, transmission of live video via the network and text generated in the local environment. The application may require a certain coherence between the two media 'text' and 'video'. Synchronization operations are needed to perform this task. In [Salm89, Stei90] the required properties were outlined. In [Salm89] some techniques for synchronization are discussed, and [Stei90] introduces the functions necessary at the API level.

3. The Application's View of Multimedia Devices

In this section we present a model of how distributed applications may control and protect multimedia devices. In essence, the model is derived from concepts of systems such as ACCENT or RSC, which support capabilities or access rights, however, only in the domain of data communications [Rash81, Eber88]. We will focus subsequently on the concept for the API which is supported by a Run Time Environment (RTE) residing in each participating node. In the examples we mainly use audio, video and illumination devices; however, the model is general and can be applied to other devices.

A multimedia workstation environment comprises many hardware devices attached to the computer. Such devices are microphones, loudspeakers, amplifiers, cameras, monitors, any kind of measuring equipment, lights, etc.. ***Devices*** may be sources and/or sinks of media streams and may allow for control operations like the zooming of a camera. Other devices may allow for control operations only, e.g. the setting of intensity and the moving of an illumination device. Also, a dedicated digital signal processor generating a sine wave may be a device. A device is located at a certain place in the system and interfaced by a device driver. The role of the device driver is to map the calls into device specific control sequences and to handle also the involved data transfer as required by the attached device.

Each device appears as an object which is described by its attributes and operations. The operations have ultimately to be supported by the associated device driver. Each device is known to the RTE by a ***device description***, which contains all information with respect to the use of the device, including addresses of device drivers and presentation information.

The actual interface to a device is via a ***capability***. A capability contains the attributes and set of operations. which are allowed to be executed by the respective application. It contains only the information required by an application.

There is only one device description per device, but multiple capabilities identifying the same device may be constructed from the respective device description or from earlier created capabilities. A derived capability is a subset of the original capability attributes and objects, which may in the extreme be the full set of operations. A device description is always held at the node, to which a device is attached. A capability may be passed to a remote node. however, the node, which owns the associated device, remains the home node of the capability. A home node may withdraw a capability. In this way, capabilities are the means, by which the owner of a device controls access to it.
Figure 2 shows the different components discussed so far.

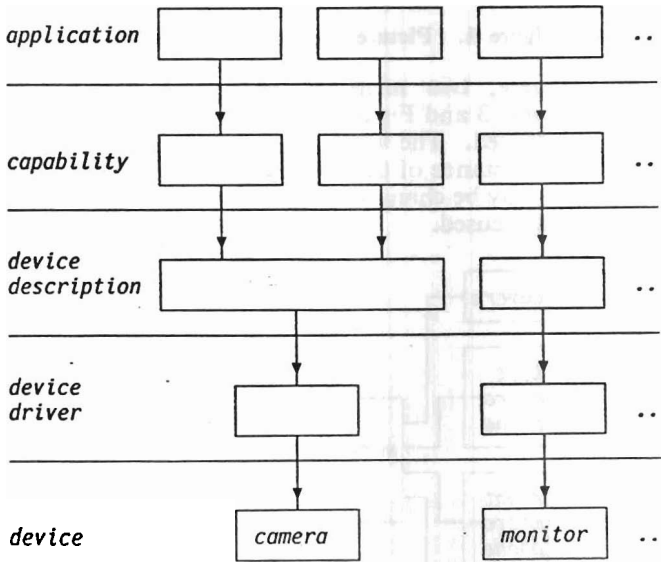

Figure 2. Example of components of the multimedia capability concept

We will illustrate our concepts by considering two interconnected workstations of which one supervises a production process via a surveillance device composed of a

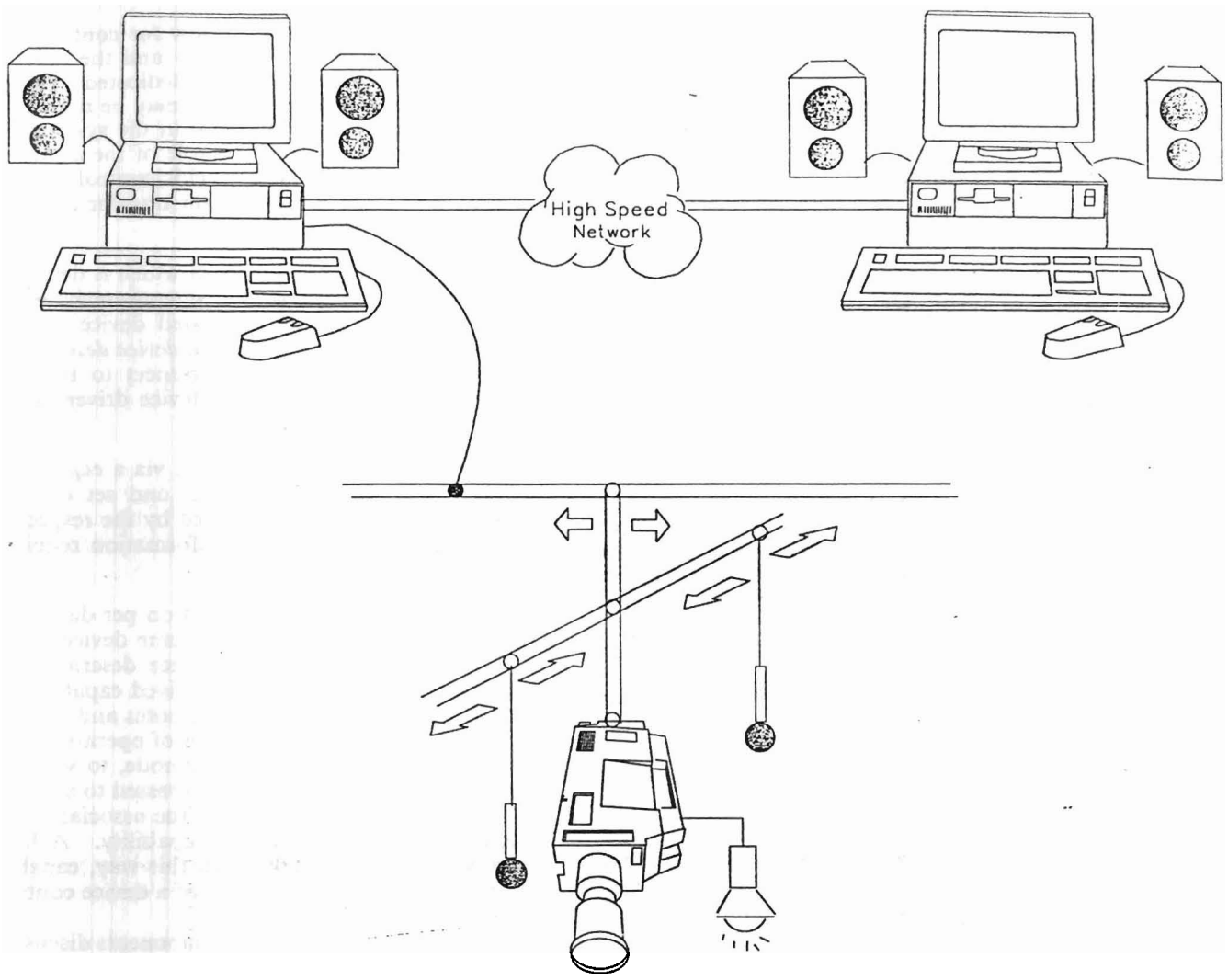

Figure 1. Picture of the surveillance system

camera, two microphones and a lamp as shown in Figure 3 and Figure 1. The intensity of the light may be regulated. The whole device may be moved in one axis. The distance of the microphones with respect to the camera may be changed. The camera may be rotated, zoomed and focused.

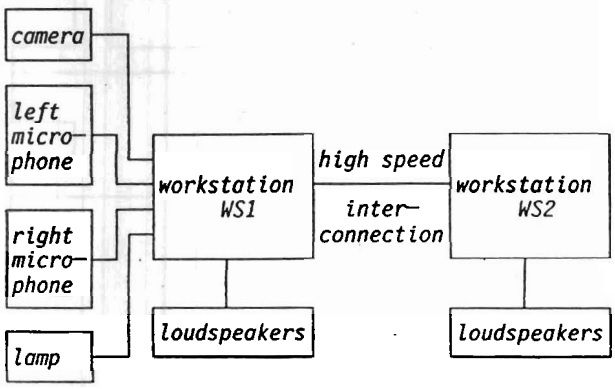

Figure 3. Schema of the surveillance system

Figure 4 contains the device description, which is named "surveyor".

```
device-description "surveyor"
  attributes
    type   : 1, "surveillance combination"
    source : 2, Video, pres=PAL,    driver=<address2>
    source : 3, Audio, pres=MASCAM, driver=<address3>
    source : 4, Audio, pres=MASCAM, driver=<address4>
    engine : 5, "illumination".     driver=<address5>
  operations
    null      1
    move      1,  0..10
    expand    1,  0..10
    rotate    2,-10..10
    zoom      2,  50..80
    focus     2,  50..120
    intensity 5,  0..100
    connect   (2,3,4)
```

Figure 4. Example of a device description

The attribute specification includes the information necessary for the capabilities. Note, that the device has several access points (1 to 5). Access point 1 identifies the device as a whole, 2 identifies the camera, 3 and 4 refer to the microphones and 5 refers to the lamp. The attribute "type" associates a description with the device (sur-

veillance combination). The "source" attribute defines access point, type of the source, and media presentation properties. Note, that the description contains information on encoding (e.g. PAL, MASCAM) which should remain transparent to the application program, but must be known by the RTE. The attribute "engine" describes the access point and properties of individually controllable parts of a device, which are neither source nor sinks but may be subject of control operations.

The operations section of the device description lists the set of operations supported by the device. Each operation has a name, is related to one or more access points, and has individual parameters. The access points used in the operations are defined in the attribute section and are used to bind an operation to a specific part of the device. For example "zoom" can be associated with the camera via the access point 2. The range of the zooming function is between 50 and 80mm. Some operations may be associated to many individual devices, as the connect operation.

However, not every operation may be bound to an attribute of the same object, the RTE will check compatibility and deny illegal or unsupported operations.

After this explanation of the device description let us discuss the example of an application running on the local environment of workstation WS1 in Figure 3, which makes use of the attached equipment. It will use the camera, the microphones and control the light. Before an application can make use of a device, it must build a capability. In contrast to the device description, which is configured in the system, capabilities are created by application programs at run time. The execution of
cap1 := make-capability ("surveyor")
creates the capability cap1. The operating system checks availability and access rights to the device before providing the capability to the application. This capability cap1 points to the device description "surveyor". Only the application which created this capability may use it.

By executing
description := view (cap1),
the description of the capability can be retrieved. The content is of the same structure as the device description except that implementation dependent information is omitted. As shown in Figure 5, the video description parameter PAL, the audio description parameter MASCAM, and the device driver addresses do not appear.

```
capability cap1 (of device-description "surveyor")
  attributes
    type   : 1, "surveillance combination"
    source : 2, Video
    source : 3, Audio
    source : 4, Audio
    engine : 5, "illumination"
  operations
    null      1
    move      1, 0..10
    expand    1, 0..10
    rotate    2,-10..10
    zoom      2, 50..80
    focus     2, 50..120
    intensity 5, 0..100
    connect   (2,3,4)
```

Figure 5. View of a capability object cap1

Now, the multimedia device and its components attached to the workstation can be programmed in the local environment using the available operations. For example, by executing
null (cap1,1),
the whole surveillance equipment is set to the home position. With
intensity (cap1,5,70),
the light is set to 70% of the maximal intensity.

As next, we consider workstation WS2 which is connected to WS1 as shown in Figure 3. Assume, the user of WS1 wants to consult the user of WS2 showing him some special parts of the production process and allowing him to adjust the camera by focusing and zooming. WS1 first constructs a capability with the desired operations by executing
cap2 := copy-capability (cap1, zoom, 2, focus, 2, connect, (2,3,4)).
The resulting capability is shown in Figure 6.

```
capability cap3 (of device-description "surveyor")
  attributes
    type   : 1, "surveillance combination"
    source : 2, Video
    source : 3, Audio
    source : 4, Audio
    engine : 5, "illumination"
  operations
    zoom      2, 50..80
    focus     2, 50..120
    connect   (2,3,4)
```

Figure 6. View of the capability object cap3 (and cap2)

Now cap2 has to be moved to WS2. It can be sent in a message or passed as parameter in a call, or it can be granted via export / import. For simplicity, we will subsequently describe the last alternative only. With the operation
export (cap2, global-name-2, user-group-A)
the capability cap2 is offered under a global name (global-name-2). The availability is restricted to the user group "user-group-A", of which we assume the WS2 user to be a member and to be accordingly authenticated.

All operations explained so far were executed on workstation WS1. On the workstation WS2, the capability must first be imported. After execution of
cap3 := import (global-name-2),
the surveillance device may be referenced via cap3 at WS2 as illustrated in Figure 7. This capability is now held at WS2, but still owned by WS1. Note that with this mechanism the autonomy of the nodes is preserved ! It is important, that cap3, which is now held at WS2, cannot be forged (e.g to contain more privileges). This is ensured by the implementation. Actually the owner holds a surrogate of cap3. At each access to the device from WS2, WS1 checks the legality of the operation against the surrogate. Illegal operations will not be performed. WS2 can now execute operations in the same way as WS1, only with the restrictions imposed by cap3. With execution of
description := view (cap3),
the attributes and the restricted set of operations available at WS2 can be viewed as shown in Figure 6. The execution of "import" as well as "view" implies communication between WS1 and WS2, which remains transparent to the application programs.

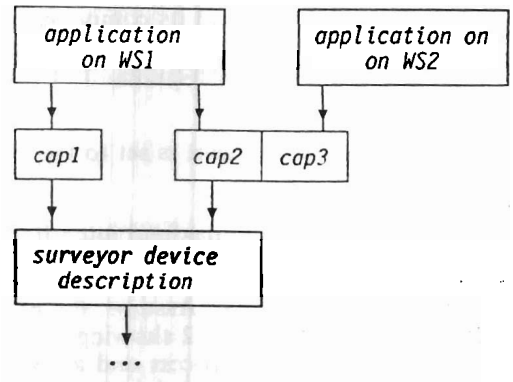

Figure 7. Capabilities cap1, cap2/cap3 of the surveillance example

The user on WS2 wants to see the production process and listen to the respective noise. His application program creates a local capability for the loudspeakers by executing

cap4 := make-capability ("loudspeakers").

A video sink is created with the execution of

cap5 := make-capability ("monitor").

The operation "connect" allows routing of streams from a source to a sink. The connection of the audio and video sources to the respective sinks is performed by the execution of the following three statements:

audio-left := connect (cap3,3, cap4,2)
audio-right := connect (cap3,4, cap4,3)
video := connect (cap3,2, cap5,2).

By executing the connect function, a connection is created, which is again an object with attributes and operations, and appears as capability (see Figure 8). The set of operations applicable to a connection may contain functions for synchronization and mixing. The very simple looking connect operation may cause a considerable amount of system handling. Apart from the establishment of a network connection with the expected quality of service, the compatibility of the devices has to be checked. Incompatible signals may have to be transformed with the respective equipment (e.g. from PAL to NTSC). The RTE of the participating workstations take care of this functionality completely transparent to the application program. Now the application program at WS2 can execute the operations on the surveillance equipment as allowed by capability cap3. For example by executing

zoom (cap3,2,90),

the zoom is set to the value of 90mm. In agreement with the requirement for distribution transparency, it makes no difference to the application program, whether the devices are located at the workstation itself or at some other place.

The above example outlined the basic features of the capability model. Objects may have further attributes and operations associated with them. Operations like snapshot, overlay, mix, or record may be applied to connections. Media storage may act as sink for persistent media information. This stored information may be used as source of other media streams. The capability model, as introduced above, is general enough to accommodate the variety of attributes and operations in the context of distributed multimedia applications.

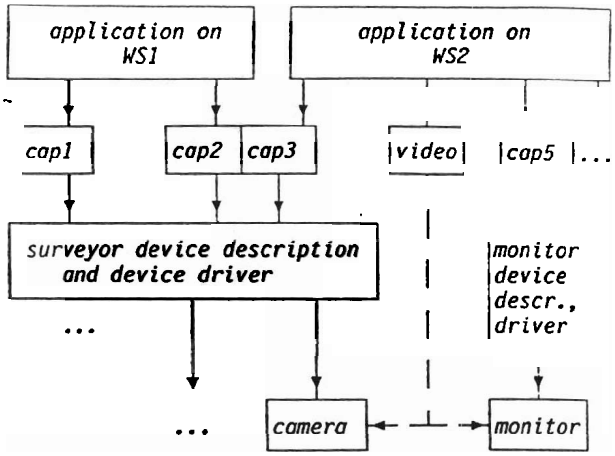

Figure 8. Some capabilities of the surveillance example

4. Conclusion

A generic application program interface for distributed multimedia applications based on the notion of device capabilities has been introduced. A device is an assembly of equipment with sources and sinks of media streams and control functions. Operations on a device are performed through capabilities pointing to a device description which in turn is linked to the device drivers.

Capability passing in distributed systems is a proven concept for combining distribution transparency with a strict access control mechanism, which is essential for mixed media cooperative processing in real time. In the proposed model, the concept of a capability is the key to the combination of the desired flexibility in program structuring with the necessary protection mechanism. The holder of a capability accesses the identified device independent of where the device is located. At the same time. the autonomy of the owner of a device or node is maintained.

By the association of descriptive information with capabilities, the development of flexible distributed applications is made possible. Programs may be written such. that they can take full advantage of the functionality of a device without being dependent on a large set of functions. This is important, since the variety of functions possibly supported is enormous and the assumption of an acceptable common subset among all multimedia attachments in a network is unrealistic.

Capability descriptions are reduced to those attributes and operations, which are necessary from an application point of view. Implementation dependent attributes and operations, such as media stream encoding expected at a source, or underlying network protocols and transfer modes are handled transparently by the system extensions to the operating systems (i.e. by the run time environment). There is no need for applications to establish network connections and to address media stream converters, mixers or synchronization devices. It is sufficient for the application to specify what it wants, the sys-

tem takes care of the rest. This is an important system contribution to the reduction of the complexity.

Multimedia communications may involve the use of devices for longer time periods. This requires the exclusive allocation of real resources, which carries with it the danger of deadlock. Classical techniques for dealing with this danger can readily be accommodated in the run time environment. Again, application programs are liberated from a burden.

This paper addressed the basic question of handling remote multimedia devices with distribution transparency and device independence. It gives an answer at the lowest level of primitives with these transparency properties. Further research is necessary to develop concepts for structuring complex multimedia applications at a higher level. For example, it may be desirable in a tutoring application, that groups of devices (for the students) can be operated upon via a single capability. This type of higher level structuring is not yet well understood. We advocate, to approach the whole area of distributed multimedia applications step by step and in a systematic way. Practical experience with operational prototypes and real applications will be very helpful in understanding the problems and the importance of their solutions to the application programmer. In this way, we hope to contribute to an understanding of the usefulness and useability of distributed multimedia applications and to the development of proper interface concepts.

5. References

[Eber88] *H.Eberle, K.Geihs, A.Schill, H.Schmutz, B.Schöner;* **Generic Support for Distributed Processing in Heterogeneous Networks;** In: G.Krüger, G.Müller (Ed.); Hector, Heterogeneous Computers together, Volume II: Basic Projects; Springer-Verlag, 1988.

[Ghaf88] *A.Ghafoor, C.Y.R.Chen, P.B.Berra* **A Distributed Multimedia Database System;** IEEE, Workshop on the Future Trends of Distributed Computing Systems in the 1990s, Sep. 1988, pp. 416-469.

[Kahn88] *P.Kahn;* **Linking Together Books. Experiments in adapting published material into Intermedia documents;** IRIS, Brown University, Technical Report Number 88-8.

[Krue88] *G.Krüger, G.Müller (Ed.);* **Hector, Heterogeneous Computers together, Volume II:** Basic Projects; Springer-Verlag, 1988.

[Leun88] *W.H.Leung, G.W.Luderer, M.J.Morgan, P.R.Roberts, S.C.Tu;* **A Set of Operating System Mechanisms to Support Multi-Media Applications;** Proc. 1988 International Seminar on Digital Communications, Zürich, March 1988.

[Leun89] *W.H.Leung, T.J.Baumgartner, Y.H.Hwang, M.J.Morgan, S.C.Tu;* **A Software Architecture for Workstation Supporting Multimedia Conferencing in Packet Switching Networks;** 2nd IEEE COMSOC International Multimedia Communications Workshop, Montebello, Quebec, Canada, Apr. 1989.

[Ludw87] *L.F.Ludwig, D.F.Dunn.* **Laboratory for Emulation and Study of Integrated and Coordinated Media Communication.** Frontiers in Computer Technology, Proc. of the ACM SIGCOMM '87 Workshop, Aug. 11-13,1987.

[Notk88] *D.Notkin, A.Black, E.Lazowska, H.Levy, J.Sanislo, J.Zahorjan;* **Interconnecting Heterogeneous Computer Systems;** Comm. of the ACM, Vol. 31, Nr. 3, März 1988, pp.258-273.

[Rash81] *R.F.Rashid, G.G.Robertson;* **Accent: A Communication Oriented Network Operating System Kernel;** Proceedings 8th Symposium on Operating System Principles, Vol.15, pp. 64-75, December 1981.

[Salm89] *M.Salmony, D.Shepherd;* **Extending OSI to Support Synchronization Required by Multimedia Applications;** IBM ENC Technical Report No.43.8904, Apr. 1989 (available through the author).

[Stei90] *R.Steinmetz;* **Synchronization Properties in Multimedia Systems;** to appear in: IEEE Journal of Selected Areas in Communications, issue on 'Multimedia Communications', Volume 8, 1990.

[Yank85] *N.Yankelovich, N.Meyrowitz, A. van Dam:* **Reading and Writing an Electronic Book;** IEEE Computer, vol.18, no.10, October 1985, pp.15-30.