GLASS: A Distributed MHEG-Based Multimedia System

H.Cossmann, C.Griwodz, G.Grassel, M.Pühlhöfer, M. Schreiber, R. Steinmetz, H.Wittig, L.Wolf[1]

IBM European Networking Center, Vangerowstraße 18, D-69115 Heidelberg

Abstract: This paper is about GLASS[2], a distributed multimedia system that is currently under development. The multimedia department of the IBM European Networking Center in Heidelberg, Germany, participates in the project along with other industrial and university partners. GLASS is an acronym for GLobally Accessible ServiceS, expressing the system's wide range of possible applications. Interactive TV scenarios can be presented as well as multimedia applications for interactive learning and games for entertainment. Clients for standard telecommunications services like FAX and E-mail can be included as well as presentation components providing access to hypertext-oriented internet services like WWW. The system's components can be highly distributed using networks with TCP/IP. An application is driven by MHEG-encoded presentations that allow for the definition of sophisticated presentations. The system comprises multiple server and client components. All components can be operated on a mix of different platforms. Although running on different operating systems, all clients have the same look and feel which is defined completely by the MHEG presentation.

1 Introduction

The GLASS project started in January 1994 and will be finished in December 1995. The goal of the GLASS project is the development of a distributed multimedia system based on the ISO/IEC MHEG standard. GLASS comprises the following components of an interactive distributed multimedia system: client systems, application server, video server and management systems. Gateways to existing services (e.g. TV, radio, World-Wide-Web, electronic mail, FAX, BTX) are currently under development.

The MHEG standardization activity is motivated by the following consideration: Standardization only at the level of monomedia information (e.g. bitmaps, text, video and audio) is not sufficient to guarantee application portability. Monomedia standardization does not address the interchange of multimedia and hypermedia information. By using the MHEG standard, the same presentation can run on different platforms providing the end user with an identical look and feel. In the GLASS project user end-systems are running on DEC Alpha, Intel, Motorola 680x0, Power and Sparc processors under the AIX, Linux, MacOS, DOS-Windows, OS/2, OSF/1 and Solaris operating systems.

---

1. E-mail: {hc,griff,grassel}@heidelbg.ibm.com, mpuehl@vnet.ibm.com, mschreib@heidelbg.ibm.com, steinmet@vnet.ibm.com, {wittig,lars}@heidelbg.ibm.com
2. This project is supported by DeTeBerkom.

This paper intends to give a brief overview of the GLASS project. It presents the major components of the system that were available at the time when the article was written. Their interaction is demonstrated by an exemplary presentation. The components allow the presentation of basic scenarios that make use of some discrete media types as well as continuous ones. User interaction is already supported but there is the possibility to add many more features to the current system. A short outlook to future extensions is given at the end of this paper.

The paper is structured in the following way: Chapter 2 introduces the MHEG standard and the MHEG Object classes. Chapter 3 describes the client and server components which comprise the MHEG run-time environment. Chapter 4 clarifies the interaction of the components by presenting a walk through the system. Chapter 5 gives a conclusion of the work presented in this paper.

2 Overview of MHEG

Primarily, MHEG is the name of the "Multimedia and Hypermedia information coding Expert Group". This group is organized as working group 12 of the ISO/IEC Joint Technical Committee 1/ Sub-Committee 29 [5].

MHEG is also the common title of the document 13552, which comprises the work of this group. The standard supports presentation, representation and manipulation modelling of multimedia and hypermedia applications. By placement with the information interchange standardization efforts, MHEG is related to the Joint Photographic Expert Group (JPEG) [2], the Moving Pictures Expert Group (MPEG) [3] and the Digital Storage Media (DSM) Group [6], which are in the same sub-committee of ISO/IEC JTC1.

This section briefly introduces the role and overall concepts of the MHEG standard.

2.1 General Concept

The MHEG standard provides an interchange format for multimedia and hypermedia information and specifies its machine-independent encoding [8]. It provides generic multimedia information structures which are suited for real-time multimedia applications, synchronization and real-time interchange of applications. Temporal and spacial relationship of monomedia objects can be expressed, timer mechanisms can be applied. User interaction with the presentation objects can be specified in detail, as well as reactions to these interactions. MHEG is also a container and description format for various monomedia formats.

To clarify the semantic information contained in a presentation coded in MHEG, a comparison to monomedia content encoding can be applied. Figure 1 depicts an analogy between the multitude of conventional text processing systems and multimedia editors. Distribution of text documents in electronic form has at the moment settled for PostScript, while in the multimedia and hypermedia domain, a multitude of players are required, one for each presentation coding. Similar to PostScript as the standard page description language, MHEG represents the common coding for multimedia and hypermedia applications. Since an MHEG presentation is not human-readable, it is also called "final form".

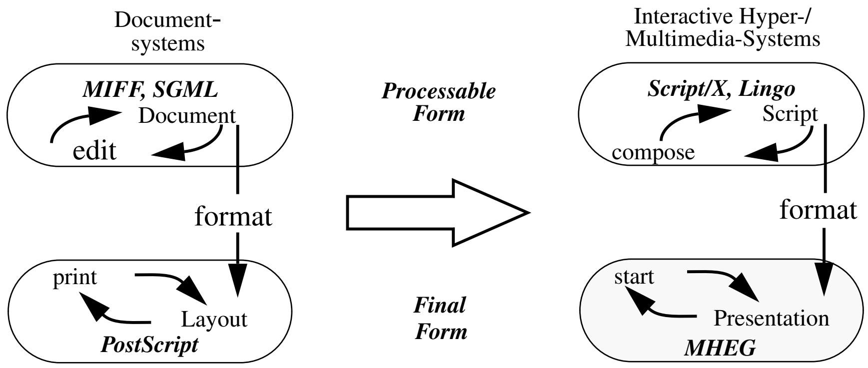

Fig. 1. MHEG as Multimedia/Hypermedia Presentation Format

The MHEG framework consists of five parts. The focus of the GLASS project is on MHEG-1 which defines the "Coded Representation of Multimedia and Hypermedia Objects (ASN.1)", and MHEG-5 which is the MHEG-1 sub-profile for interactive and digital television. The experiences of the MHEG system design and implementation within the GLASS project has influenced the ISO/IEC MHEG-1 and MHEG-5 standardization activities.

2.2 MHEG Classes

MHEG makes an object oriented approach towards presentation interchange. It describes a presentation by means of class descriptions. From these descriptions, classes may be set up in the MHEG interpretation service to form so-called interchanged objects, which can in turn be instantiated to run-time objects. Figure 1 shows

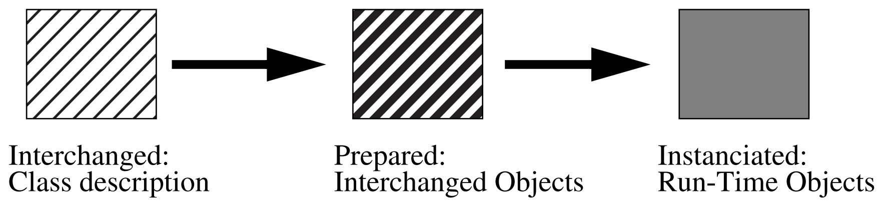

Fig. 2. Object-Oriented Modelling

the three steps that are required by the MHEG interpretation service to create a run-time object. An effect of this modelling principle is that presentations can not be started by the presentation provider, but that the interpretation service must always take the initiative and retrieve at least a single initial object to start a presentation. This start-up mechanism is not described in the MHEG documents and is private to each MHEG presentation system.

Content Class

The content class refers to or contains the coded representation of mono-media information together with a set of parameters containing information that is required for the presentation of content. This set contains information on the coding method and a specification of the application-oriented parameters (e.g. color table, Quality-of-Service parameter). They form the description of object classes that are set up in the MHEG Engine as Interchanged Objects.

Multiplexed Content Class

This class refers to or contains the coded representation of multiplexed media data together with a description of each multiplexed stream.

Container Class

The container class provides a container for grouping multimedia and hypermedia data in order to interchange them as a whole set.

Composite Class

The composite class provides support for multimedia and hypermedia objects that are processed as a single entity. This mechanism provides a consistent approach to the synchronization in time and space. It also provides short-cuts by allowing action objects to be applied to groups of objects.

Action Class

The MHEG standard defines an initial behavior for each MHEG Object. It also describes how to modify the initial behavior of each object by defining a list of elementary actions to be applied to the objects. The actions are used within a link object to describe a link effect. The MHEG standard defines the following types of behavior:

- Preparation: loading of objects into the system and removing them (Prepare, Destroy)
- Creation: instantiation and deletion of run-time objects (New, Delete)
- Presentation: control of the progress of the run-time objects (Run, Stop)
- Rendition: control of the projection of the run-time objects (e.g., Set Speed, Set Size)
- Interaction: control of the result of an interaction (e.g., Set Selectable, Set Modifiable)

Actions are also provided to retrieve a current value of an object's attribute for further processing in Links or Actions.

Link Class

The Link Class defines a structure which specifies a set of relationships. Each relationship is defined between one or more sources and one or more targets. The relationship is composed of conditions associated with the sources (link condition) and the actions to be applied to the targets (link effect). The actions are to be applied to the targets when the conditions become true. An exemplary link object is shown in Figure 1.

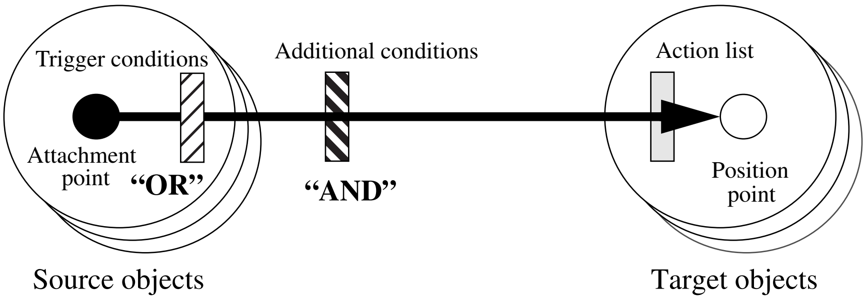

Fig. 3. Link Class

Script Class

The script class defines a container for complex relationships between MHEG Objects, defined by a language not defined within the first part of the MHEG standard. Within the standardization body, the MHEG-3 effort is undertaken in order to provide an exchange format for the scripting language.

Descriptor Class

The descriptor defines a structure for the interchange of resource information about a single object or a set of multiplexed objects. For example, this information is used to facilitate a correspondence between the resources required to present the objects and the resources available to the system.

3 Architecture of the GLASS system

The GLASS system consists of various components that work together to make an MHEG presentation run and to provide the user with visual and audible output as well as with devices for interaction. The components can be distributed over different machines on interconnected networks and they are able to exist in a heterogeneous environment. This is possible because they use project specific application protocols that are based on a standard internet protocol suite that includes TCP/IP. A couple of the protocols will be described along with the components that communicates with each other.

The components can be subdivided into three categories:

- Client Components, consisting of the User Interface Agent (UIA), Presentation Objects (POs), the MHEG Engine (Engine) and the Control Agent (CA).

- Management Components, consisting of the Session Management Agent (SMA) which, in turn, includes the Authoring Agent, the Security Agent, the Locator Agent, the Directory Agent, and the Data Distribution Agent.
- Stores consisting of MHEG Object Stores and Content Data Stores.

Figure 4 gives an overview of the system´s architecture.

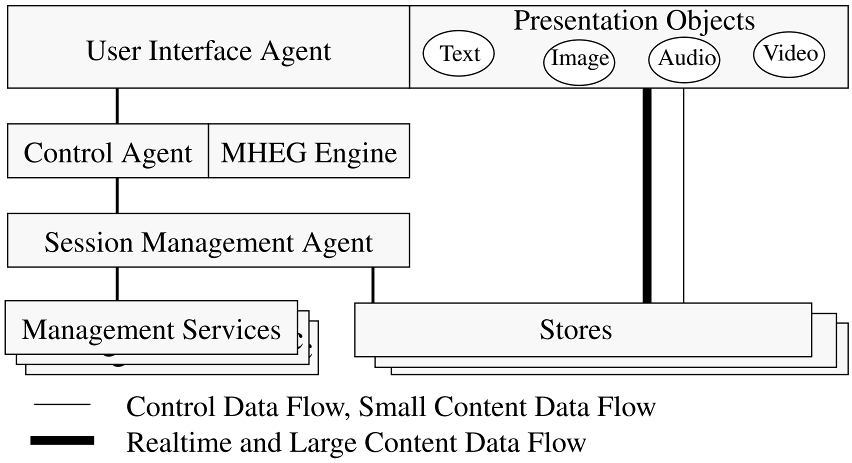

Fig. 4. Architecture of the System

The User Interface Agent is responsible for the management of Presentation Objects that autonomously present the audible, visible and interactive parts of a presentation. The MHEG engine is the heart of the system. It interprets MHEG presentation objects and drives the UIA and its POs. It's communication front-end is the Control Agent. It is connected to the Session Management Agent that manages subcomponents for data distribution, security and accounting purposes. Stores are components that are responsible for content delivery of either discrete or continuous media. The components will be described in more detail in the following paragraphs.

3.1 Client Components

User Interface Agent

The User Interface Agent (UIA) is a client component that is responsible for the creation, maintenance and destruction of Presentation Objects. It is completely driven by the MHEG engine with which the UIA communicates through an intermediate component - the Control Agent. Communication with the Control Agent is done using the User Interface Control Protocol (UICP). This protocol has primitives for session establishment, control and accounting, as well as primitives for Presentation Object control. A typical transaction consists of a Request that is asynchronously answered with a Response. Most requests originate from the MHEG engine that wants the UIA to perform another presentation task. Most of the UICP PDUs (Protocol Data Units) represent atomic MHEG actions that are combined to perform the presentation. Nevertheless there are PDUs from the UIA that either request an SMA service or signal an event to the engine. The Control agent routes these accordingly.

The UIA design is thread-based to allow for concurrent Presentation Objects that will act independently from each other without blocking the UIA from its management tasks. Figure 5 shows the UIA's architecture.

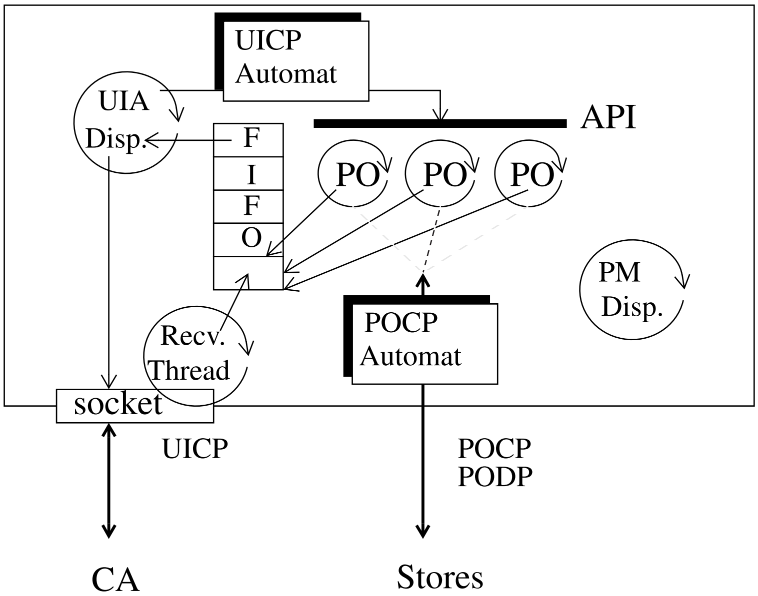

Fig. 5. Architecture of the UIA

All incoming and outgoing PDUs are collected into a queue that guarantees the correct execution sequence. A protocol automaton is used to build an internal object oriented representation of the MHEG presentation´s elements that are currently in use. A set of Presentation Interchange Objects (PIOs) for atomic presentation elements like JPEG Images, MPEG Video, MPEG Audio and Text is constructed. PIOs allow for efficient resource management to prevent the POs from multiple buffering of content data. They also reflect MHEG´s concept of presenting contents in two steps. The first step is used to prepare POs for running, e.g allocating resources, retrieving data and initializing devices. It is completely transparent to the user and ensures that the prepared PO is ready to run. The second step starts the presentation of a PO that then becomes visible (or audible) to the user. The UIA only tells the appropriate PO to start its presentation, the presentation process itself is out of the UIA's scope and up to the PO. When the MHEG Engine tells the UIA to do so, a PO or a PIO with all POs belonging to it will be deleted from memory.

Presentation Objects

Presentation Objects (POs) handle the presentation of content data objects as well as the user interaction with these objects. POs are created, modified and destroyed by the UIA on demand of the MHEG Engine (Engine). Content data objects, managed by

Presentation Objects, are either extracted from an MHEG Object, if included, or retrieved from the Content Store, if referenced in the MHEG Object. The Presentation Objects handle the necessary communication in the latter case. They communicate with the store using the Presentation Object Control Protocol (POCP) and the Presentation Object Data Protocol (PODP). The POCP establishes and controls the PODP which performs the actual data transmission. While the POCP provides primitives such as "open data connection", "start/stop data streaming", "set stream speed" etc., the PODP transmits raw data of different media types.

Presentation Objects are responsible for presenting multimedia data to the GLASS system's user. Besides, for interaction purposes, POs map events generated at the proprietary graphical user interface (e.g. mouse and keyboard actions) into a form that the Engine can interpret. The Engine is able to respond to these events in a way the presentation author has intended, for example it allows the user to navigate through the presentation.

Media types like audio and text are supported by different types of POs. This allows for easy extension of the GLASS system. Currently supported media types are

- Video (MPEG I),
- Audio (MPEG-Audio and WAVE),
- Images (JPEG) and
- Text (plain text and a GLASS text format)

which are associated with the respective POs. This set of POs can be separated into two subsets of POs according to the contents' media types: discrete and continuous. Video and audio are examples for continuous media types. They may neither be cached nor instantiated more than once at a time from an interchanged object. They require real time flow-control and are played out directly on the local system. Except for this, they work like discrete media. Examples of those discrete media that can be preloaded when the PO is instantiated are pictures and texts. The GLASS architecture for example allows the ImagePO to receive an entire JPEG image, to decode it and to store the decoded image for other ImagePO instances' usage.

MHEG Engine and Control Agent

The MHEG Engine (Engine) is the driving and controlling force of MHEG presentations. Its task which is central to the system is to control the presentation. To fulfil this task, the MHEG Engine interprets the interchanged MHEG Objects. As a result, it issues requests to the presentation system.The interpretation process might also result in requests for retrieval of further objects.

The Engine requests MHEG Objects asynchronously from the Control Agent (CA) which in turn makes use of the MHEG Stores through the Session Management Agent to retrieve these objects. The result of this retrieval process is an asynchronous response as an input to the Engine. Another source of input for the Engine are events. These events originate in the presentation system and are passed to the Engine through the User Interface Agent and the Control Agent. MHEG Objects and events are processed by the Engine. This may result in a request to the presentation systems or trigger the retrieval of further MHEG Objects. Internally, the Engine manages a single

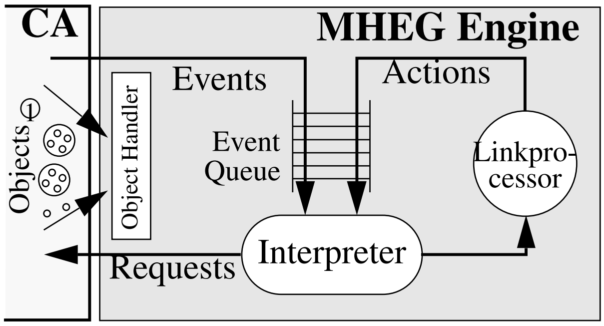

Fig. 6. Architecture of the MHEG Engine

event queue which serializes the events received from the presentation system, the notifications of MHEG Objects provided by the CA, and actions that are triggered by internal state changes which are the result of processing such incoming information. Actions will in turn result in the retrieval of new objects, or sending of requests to the presentation system.

The Control Agent (CA) is a service instance which handles all communication with remote or local system components for the MHEG Engine. During a session setup phase, which is always initiated from the client side, the Control Agent establishes a connection to the Session Management Agent and initializes the MHEG Engine. Afterwards, it processes PDUs from the three components according to the automaton described in [1]. Packets can be either translated to another protocol and forwarded or they can be dropped.

As a switching component, the CA is connected to three other components, and it requires up to three different communication mechanisms. While the connection to the SMA is always networked and uses TCP sockets, and the communication with the MHEG engine is always local and uses message queues, the connection between the CA and the UIA uses either of them depending on the way the UIA connects to the CA.

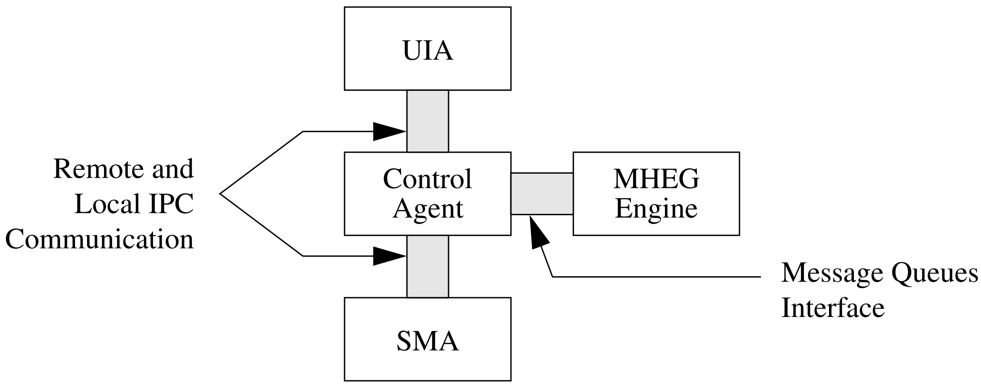

Fig. 7. Control Agent's Interfaces

Besides handling the communication for the Engine, the Control Agent services the timer interrupts required for the timer management of the Engine. For a distributed environment, the Control Agent is implemented to work on both client and server sides of the system. Basically, putting the CA and Engine on the client achieves better performance, while putting them on the server reduces resource requirements on the client system and achieves greater flexibility. This flexibility has major effects on the design of the component, in that it must allow an efficient implementation on the clients' operating systems that can also be applied to a server operating system without major changes.

The state information held in the CA does not contain information about the activity state of a particular presentation or presented object. Rather than that, the state information is only concerned with the connection itself. Because of this, messages can arrive during the run of the session from all three neighboring entities in no particular order as far as the knowledge of the CA is concerned.

In order to service all different kinds of connection, the CA consists of two threads, one of which exists initially. These two processes have no need for communication except for common knowledge of the state of the CA automaton. Thus, this state is kept in shared memory and accesses to the state variable are protected by semaphores.

At start-up time, only one thread, the CA-Server-Thread, is active. The MHEG Engine is uninitialized. When a UIA connects to this server and sends a Connect Request message, the CA-Server-Thread establishes a connection to a Session Management Agent (SMA), initializes the MHEG Engine and splits into two threads, namely the CA-Sender-Thread and the CA-Receiver-Thread. It is advisable that the MHEG Engine runs as another thread in the same process as the CA, allowing for message passing in shared memory.

3.2 Server Components

Session Management Agent

The Session Management Agent (SMA) organizes the retrieval of MHEG Objects and content data objects using the Locator Agent, Accounting Agent, and Security Agent. The Control Agent requests MHEG Objects from the Session Management Agent that retrieves these objects from the MHEG stores and passes them back to the Control Agent. This mechanism is not used for the transfer of referenced content data objects but only for MHEG Objects and control information. The referenced objects are transferred between the Content Data Stores and the Presentation Objects by individual transport mechanisms. However, the transport is initialized by the Session Management Agent.

The communication between SMA and Stores is performed using the Store Control Protocol (SCP). This protocol contains primitives for the initiation of data transfers between Presentation Objects and Stores/Gateways, for the retrieval of MHEG Objects from MHEG Stores, and for accounting purposes.

There are several subsystems and sub-agents in the SMA that will not be discussed in this document.

Content Store

The Store is responsible for the storage of content data and the transmission of this on request towards its clients. As such, it provides:

- mechanisms to control the exchange of content data with Presentation Objects,
- transport system mechanisms for transfer of continuous-media data.

In later stages of the implementation the Store will offer additional transport system mechanisms for transfer of continuous-media data with varying transmission characteristics depending on the Quality of Service (QoS) requirements of clients, functionality to pass accounting information back to the Session Management Agent and protocols for the communication with Locator Agent and Data Distribution Agent.

The Stores communicate via exchange of PDUs with Presentation Objects (POs) and the Session Management Agent (SMA).

The Store consists of two components (Figure 4):

- a Video on Demand (VoD) Server including appropriate stream handlers (SH) to perform the handling of multimedia data, i.e., the storage and transmission of continuous-media data, and
- a Control Process (*ctrlproc*) which communicates via GLASS protocols with other GLASS system components (i.e., POs and SMA) and instructs the *VoD server* accordingly to deliver the required multimedia data stream.

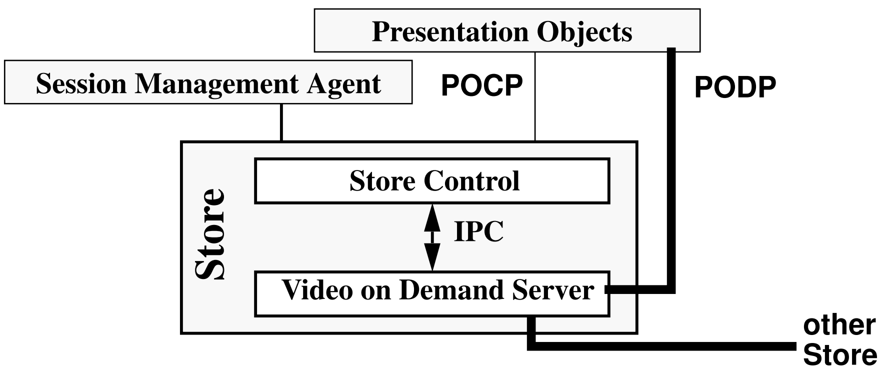

Fig. 8. Store Interfaces

Control Process

The *ctrlproc* consists of and uses various subcomponents:

- Communication Interface to exchange information with other GLASS components
- Handler modules (pocphdlr, scphdlr, credhdlr) for the control protocols POCP and SCP,
- Peer, the internal representation of a peer, i.e. the receiving Presentation Object on the receiving side,

- Key, the internal representation of an SMA key,
- cm_stream, the internal representation of a content data stream transferred to a Presentation Object
- server_conn, the module which controls the communication with the *VoD server*

To set up a stream, two prerequisites are required. Firstly, the SMA must provide a key to the control process, secondly, the peer must authenticate itself with this key for content retrieval. If this authentication is successful, the stream is opened. Further requests from the peer are required to start streaming of data.

Processing within the *ctrlproc* is done within one thread. Since it is not responsible for the transmission of large content data such as continuous-media data but performs control operations only, this incurs no performance drawback and simplifies implementation and testing.

Video on Demand Server

The Video on Demand server used within GLASS is a server designed for IBM RISC System/6000 workstations running under IBMs AIX Version 3 operating system. To provide flexibility, the server allows dynamic addition of hardware- and protocol-specific stream handlers.

The server is designed to provide high quality, guaranteed, on-time delivery of continuous-media data such as video, audio, or animation from a server to desktop computers located around an existing network. It allows delivery of *streams* of multimedia data (e.g. video) to desktop computers for applications wishing to “play” multimedia data. At the same time it provides non-real-time access through existing network filesystem interfaces and maintains concurrent use of the workstation for standard applications. This is possible by guaranteeing smooth play-out using resource reservation mechanism as described in [9].

The filesystem can determine whether it can meet the needed quality of service for a specific stream before the stream is started. It ensures that it can meet these requirements by tracking the complex interactions of multiple simultaneous streams with different qualities of service.

It provides a format that is optimized for the storage and retrieval of multimedia data while traditional filesystems are optimized for the storage of small data files.

This is enhanced by a set of generic multimedia services and resource reservation for UNIX systems. It utilizes the facilities provided by the filesystem to provide an integrated continuous-media server and to provide generic resource control for the various subsystems. They also provide interfaces to easily load new modules to support different types of networks for customers with special requirements.

4 Interaction of the components

The last chapter introduced the components of a GLASS system in a very abstract manner. This one wants to clarify their functionality and the communication among them from a more practical perspective. A very simple presentation will be used for a walk through the system.

The elements of a simple presentation are shown in Figure 4. The first page con-

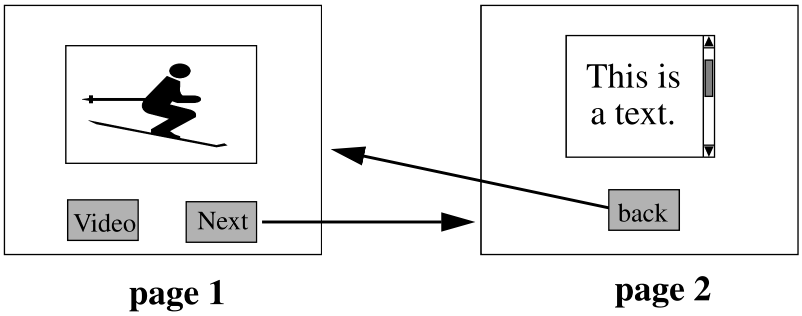

Fig. 9. Simple Presentation Example

sists of a background image and two small images labelled ‘Video’ and ‘Next’. Both labelled images are defined to be selectable, which makes them buttons. In the center of the background image is a frame that is used to overlay a video clip. If the user presses ‘Video’ the video clip will be displayed in the overlay area. Clicking the ‘Next’ button will lead to page 2.

Page 2 consists of a background image that is overlaid with a smaller text window. The user can scroll through the text using a scroll bar and up and down buttons. A small image labelled ‘back’ is positioned below the text window. Button functionality is assigned to it. If it is clicked it leads back to page 1.

Now let us start a walk through the components. All MHEG Objects that belong to the presentation are stored in an Object Store. All contents, the discrete as well as the continuous, are stored in a Content Store. We assume that the SMA is set up to receive requests (e.g. running or automatically starting by the inetd) and that the MHEG Engine is waiting for our presentation’s initial object. Figure 4 shows the connection

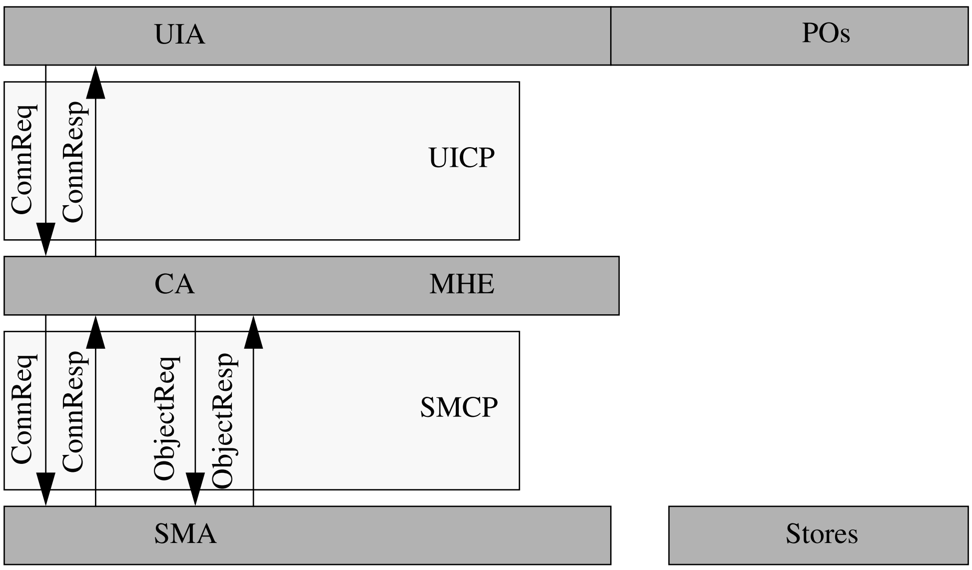

Fig. 10. Communication among the components: startup

setup as it is requested by the UIA. When the user starts the UIA, e.g. by clicking on its icon on the client machine, the presentation begins. A UICP Connect Request PDU is sent from the UIA to the CA. The CA forwards the request to the SMA and responds to the UIA after the connection between SMA and CA is set up. Then, the CA retrieves the initial object and feeds it into the Engine which starts interpreting it.

We assume that in the described case, the initial composite object contains all MHEG Objects required to present the first page. Any further object requests from the engine would be satisfied by the CA that communicates with the SMA using the SMCP protocol. With each object request, the SMA checks from which store the MHEG Objects are available and retrieves them. The retrieval mechanism and the communication between the SMA and the media stores are out of the scope of this document.

While interpreting the initial composite, the MHEG engine is usually required to prepare monomedia contents that are referenced in the presentation. A presentation without monomedia content does nothing user-detectable, and a presentation consisting purely of included content is not feasible for a retrieval system like the GLASS system. It is advisable to transport small data portions as included data because of the transportation overhead, whereas big data quantities should be transmitted as referenced data because all included content data is copied into the MHEG engine once.

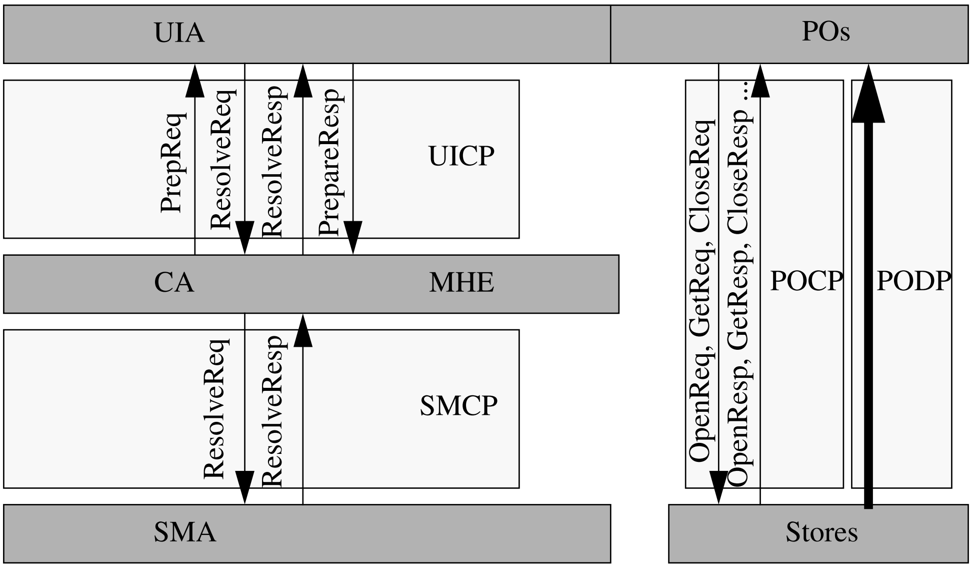

Fig. 11. Communication among the components: content preparation

Figure 4 shows how the MHEG engine uses the UICP primitive PrepareRequest to tell the UIA which Presentation Object is to be prepared for execution. In our example a PrepareRequest PDU for each of the three images on the first page is sent to the UIA. The UIA builds internal representations for these objects that are called Presentation Interchange Objects (PIO). Such a PIO has to communicate with the SMA for resolution of the references to monomedia content (ResolveReq and ResolveResp primitives). The resolution is provided as a uniform resource locator (URL) indicating the protocol required (POCP), machine name, optional port number, and an additional string resembling a Unix-like path to identify the content.

The PIO then uses the POCP protocol to build a connection to the store that is pointed to by the URL's machine name. When a connection has been established the PODP protocol is used to retrieve the data. As content is stored and transferred in a standard format, e.g. JPEG for images, it is necessary to convert it to the client machine's local content type. After this is done, the UIA answers the successful preparation of the objects with a PrepareResponse.

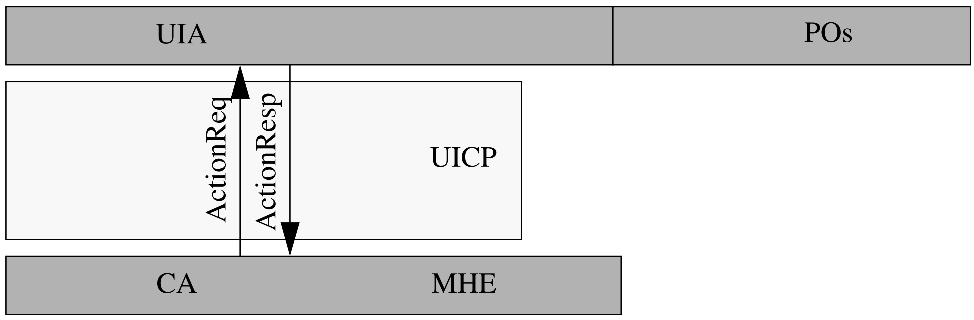

Fig. 12. Communication among the components: actions

After the prepare phase has finished, the engine can execute actions that are instances of the Behavior Class on the prepared objects. An action is considered finished after it has sent an ActionReq PDU to the UIA, been synchronously executed there, and an ActionResp PDU has been sent back to the engine as shown in Figure 4. The first action sent to a Presentation Interchange Object (PIO) has to be the New action. For each prepared image, one New action is sent from the engine to the UIA. The UIA creates for each New action a Presentation Object (PO). Each PO derives from a PIO whose content it inherits and shares with optional other instances of the same PIO. Thus the images have to be stored in the client's address space only once. After instantiation, a PO is ready for presentation to the user.

The MHEG engine now instructs the UIA to perform a Run action on each Presentation Object to show it on the screen. In parallel the engine interprets some Behavior objects that enable the images labelled 'Video' and 'Next' to be selectable and installs some Link objects that start video playback and creation of the second page, respectively, when one of the (now selectable) images is clicked. These are actions propagated to the UIA using the UICP protocol as well. The UIA itself tells the two POs to be clickable and to send back a message when being clicked.

When the user clicks the 'Video' button, the UIA informs the engine of the event. The engine knows that the button click triggers the video playback and sends a Run action on the video Presentation Object to the UIA. A video clip is a special type of referenced data object, a continuous one. It can not be downloaded in a burst like an image. The control (POCP) and data (PODP) connections between the store and the PO as presented in Figure 4 have to remain open during the existence of the video PO. Data is transmitted continuously as a real-time data stream until the end of the clip is reached or another event in the presentation results in preliminary destruction of the video PO. Then the connection has to be closed. In our simple example the user has to

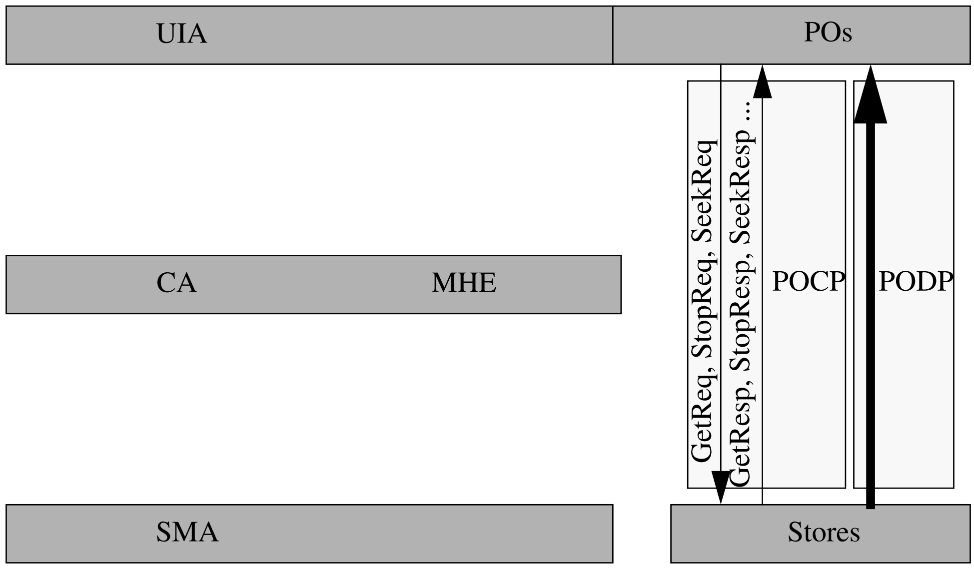

Fig. 13. Communication among the components: video playback

wait for the end of the video clip until he is able to interact with the presentation again. Nevertheless, it is possible to express a VCR like functionality in an MHEG presentation using the SetSpeed and SetPosition actions.

Clicking the 'Next' button will lead to the second page. The MHEG engine will (in our example) clean up all resources used by the first page and will instruct the UIA to do so as well. The Stop action is used to hide a Presentation Object from the screen. The Delete action is the counterpart to the New action and will delete a Presentation Object. The Destroy action cleans up all resources that were allocated with the Prepare action.

The second page is created in the same way as was explained for the first page. Clicking the 'Back' button will lead back to the first page that has to be created again as described above. This procedure can be sped up considerably by not destroying downloaded contents in a page transition that are expected to be reused. As we have seen our simple presentation example is a kind of endless loop. It can only be ended by stopping the UIA. In that case, a DisconnectReq PDU is sent to the CA that answers with a DisconnectResp PDU. After that the MHEG engine is re-initialized and waits for a new connection from the same or another UIA to start the presentation again.

5 Conclusion

The following chapter summarizes the results of the paper and gives an outlook on future extensions to the system.

5.1 Summary

Before coming to a conclusion about the results of the GLASS project as presented in this article, it may be helpful to think about the situation of today's multimedia applications.

Today there are only few distributed multimedia systems available. Most of the systems are intended for running on a single system, though multiple platforms are supported. These are kiosk systems and presentation systems for Point-of-Information/ Point-of-Sale scenarios. There are a lot of multimedia authoring systems available that allow user friendly and easy construction of those applications. However, there is a lack of interoperability and interchangeability of the presentations among the presentation systems. Thus, content providers have to decide which systems they want to support and which to ignore. This decision will be risky for them as well as unsatisfactory for the user in the consumer market because only a subset of applications are available for him depending on his decision for a system.

Those limitations can be overcome by sticking to an international standard such as MHEG. In this sense GLASS is a pioneer project. It makes use of MHEG's concepts and implements an engine that is capable of interpreting MHEG Objects as defined in the MHEG DIS document. On the other hand the MHEG standardization process from DIS to the final IS profits by the GLASS designers' experiences. So the GLASS project and the MHEG standard mutually influence each other.

The GLASS architecture defines a distributed system that allows coexistence of system independent components as well as system dependent components that communicate using protocols based on TCP/IP. System independent components can be easily ported to different platforms and system dependent components are shielded to allow a heterogeneous system. Since Presentation Objects make use of standard media types like JPEG images and present them without applying features that are specific to a graphical user interface, the same look and feel is maintained across all platforms.

Coming to a conclusion one can say that the GLASS system is demonstrating the successful use of a standard multimedia and hypermedia exchange format like MHEG to guarantee both the content provider's and the end user's investments without losing flexibility.

5.2 Future Outlook

The GLASS system as described in this article provides all the basic means for a flexible and extensible multimedia system. These means can be used in many ways to support the user with global services that really demonstrate the power of the architecture.

Currently under development are gateways that build interfaces to standard services like TV and radio broadcast, BTX, World Wide Web (WWW) and Electronic Mail. Some gateways make use of facilities that allow real-time compression of continuous media using standards like MPEG. Others convert hypertext information and semantic actions to MHEG Objects that are used to build MHEG presentation parts dynamically.

Another important task will be to develop tools that allow designers to quickly and easily produce MHEG-based multimedia presentations. Those tools would help to increase the number of available presentations rapidly. Until then presentation designers could be provided with converters from other multimedia authoring tools to MHEG.

Since the MHEG standardization is still in process, it will be necessary to modify the engine when the official MHEG International Standard document is passed.

As outlined, there is still some work to do but the existing GLASS system provides a good starting platform for further research activity.

Acknowledgements

The authors thank all partners of the GLASS consortium, namely DEC CEC, DeTeBerkom, GMD Focus, Grundig Multimedia Solutions, TU Berlin PRZ for the co-operative work. Many thanks to DeTeBerkom for sponsoring and supporting the project. The authors express their gratitude to Hans Werner Bitzer, Thorsten Illies, Stefan Koenig, Thomas Meyer-Boudnik and Olaf Rehders for their discussions and contributions.

References

1. DEC, GMD Fokus, Grundig Multimedia Solutions, IBM ENC, Technical University of Berlin PRZ: *BERKOM Globally Accessible Services: System Specification 1.0.* BERKOM, Berlin, May 1994.

2. ISO/IEC IS 10918:1992: *Information Technology – Digital Compression and Coding of Continuous-Tone Still Images* (*JPEG*). 1992.

3. ISO/IEC IS 11172:1992: *Information Technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s (MPEG).* 1992.

4. ISO/IEC IS 11544:1992: *Information Technology – Digital Compression and Coding of Bi-level Images (JBIG).* 1992.

5. ISO/IEC WD 13818-6:1994: *Information Technology – Coded Representation of Multimedia and Hypermedia Information Objects (MHEG) – Part 1: Base Notation (ASN.1).* June 1993.

6. ISO/IEC CD 13552-1:1993: *Information Technology – MPEG-2 Digital Storage Media Command and Control Extension (DSM-CC).* November 1994.

7. ITU-T Draft Recommendation T.170: *Audiovisual Interactive (AVI) Systems – General Introduction, Principles, Concepts and Models.* Second Revision, Geneva, CH, 16-25 November 1993.

8. T. Meyer-Boudnik, W. Effelsberg: *MHEG - An Interchange Format for Interactive Multimedia Presentations.* Accepted for IEEE Multimedia Magazine, 1995.

9. Carsten Vogt, Ralf Guido Herrtwich, Ramesh Nagarajan: HeiRAT: The Heidelberg Resource Administration Technique - Design Philosophy and Goals. Tagungsband Kommunikation in Verteilten Systemen, Munich, Germany, March 3-5, 1993