REIHE INFORMATIK
7/94

MHEG: An Interchange Format for interaktive Multimedia Presentations

Thomas Meyer-Boudnik, Wolfgang Effelsberg
Universität Mannheim
Seminargebäude A5
68131 Mannheim

accepted for IEEE Multimedia Magazin 1995

MHEG:
An Interchange Format for Interactive Multimedia Presentations

Thomas Meyer-Boudnik, Wolfgang Effelsberg
Universität Mannheim
{mey, effelsberg}@pi4.informatik.uni-mannheim.de

Abstract

This paper surveys the upcoming MHEG standard, as under development by the Multimedia Hypermedia Experts Group of ISO. It's scope is to define a system-independent encoding of the structure information which can be used for storing, exchanging, and executing of multimedia presentations in final form. We use a running example to describe the building blocks of an MHEG encoded presentation. To contribute our experience using MHEG in a networked multimedia kiosk environment, we present a complete MHEG run-time environment.

1 Introduction

The multimedia market is characterized by a large number of manufacturers, each one using their own technology for digital audio and video, for script languages, for encoders/decoders, communication protocols, operating system extensions etc. There is currently no standardized way to compose a complex and interactive multimedia presentation, store it on a medium, send it over a network and play it out on somebody else's computer. The situation is comparable to networking in the late 1970s when every manufacturer had his own networking architecture, and heterogeneous computers could hardly interoperate. In the networking field the definition of protocol standards has created a tremendous potential for the interchange of information between all kinds of computers world-wide. It is expected that the creation of open standards for multimedia presentations can have a similar impact.

Today there is a strong interest on the part of the information providers to create multimedia presentations, e.g. books enhanced by audio annotations or short video clips. Many joint ventures were created between computer companies and major publishers in order to get actively involved in this market. But it is still unclear today in what document architecture an author should create his document in order to be as portable as possible. No single system has penetrated the world-wide market and established a de facto standard. For example, a multimedia book would have to appear with several video formats, e.g. Compact Disk Interactive (CDI, MPEG-1) or Apple's QuickTime. This is inconvenient and expensive.

Similarly a typical multimedia end user would have to own several different systems today in order to be able to view all the available presentations: a PC, an Apple Macintosh and a UNIX workstation would be the most important ones. Those having only one system are not able to access all the available information. To con-

clude, there is a strong desire on both the multimedia authoring side and the multimedia consumer side to define a universal standard and implement it on all major platforms.

Multimedia technology is now readily available on many computer systems, in particular PCs and UNIX workstations. Storage and playback is typically done locally, without transmission over networks or exchange of data between heterogeneous systems. The reason is that there is a limited use of international standards for representing multimedia contents, and in particular for the representation of conditional links, and spatial and temporal relationships between such contents in final form (multimedia script). This problem has been addressed by the International Organization for Standardization (ISO) in subcommittee ISO/IEC JTC1/SC29 (Coding of Audio, Picture, Multimedia and Hypermedia Information). The subcommittee has three working groups: one on JBIG/JPEG, one on MPEG, and one on MHEG, as shown in Figure 1.

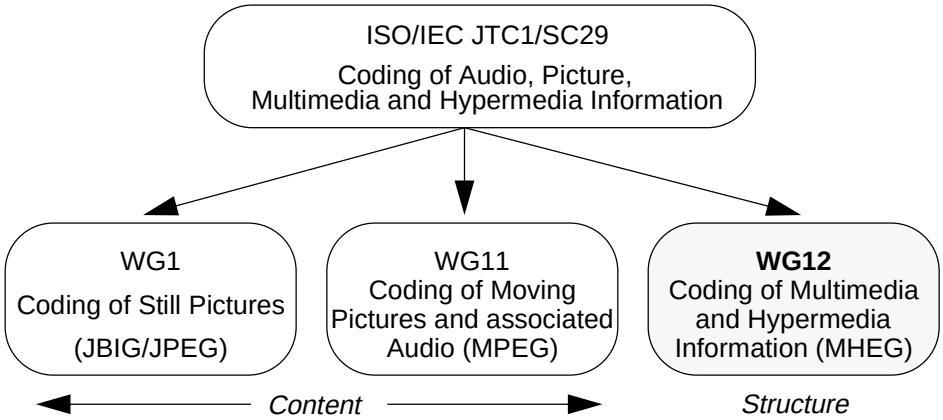

Figure 1: ISO working groups in subcommittee 29

The JPEG Standard defines compression algorithms for still images [ISO10918]. These compression algorithms are sometimes also used for video; this is often called “Motion JPEG”. The main disadvantage of this approach is that the compression algorithms cannot take into account the temporal redundancy in a video stream since JPEG compresses each frame in isolation. Therefore, the MPEG Standard was created, which defines a compression and interchange format for motion pictures, including audio [ISO11172]. Both JPEG and MPEG are well known and widely used in multimedia systems.

The JPEG and MPEG standards both describe the contents of information objects only. There are no means in these standards to describe the interrelationship between the different pieces of a multimedia presentation. The definition and standardization of such structure information is the purpose of the *Multimedia and Hypermedia Information Coding Experts Group* (MHEG).

Previous publications about MHEG have presented the scope of the standard [KrCo92, Pric93]. It is the purpose of this paper to present the basic ideas of MHEG in more detail. We first introduce the MHEG model in Section 2. MHEG is object-oriented, and in Section 3 we present the MHEG classes and their relationships. We use a running example for each class in a slightly simplified ASN.1 notation. The evolution of the MHEG model from the CD to the DIS version is discussed. Section 4 focuses on implementation issues: how an MHEG presentation is created, how it is transmitted, and how it is executed in an MHEG run-time environment. Section 5 concludes the paper.

2 MHEG Concepts

2.1 MHEG's Data Interchange Model

Although MHEG is an interchange format, it is more than only a binary format. It also comprises a number of features that make it suitable for real-time interchange in a networked environment. Before we describe these features, we introduce the several levels of representation: classes, objects and run-time objects as parts of MHEG's data interchange model.

MHEG Classes
Since MHEG is part of ISO, the data interchange model follows the ISO philosophy, as defined in the Basic Reference Model for Open Systems Interconnection (OSI). In particular the encoding of MHEG is based on the architecture of the ISO presentation layer. A key concept here is the separation of abstract syntax and transfer syntax: In the abstract syntax two communicating application entities describe their data elements and data structures, defining their "universe of discourse". In this context ISO has defined a notation called Abstract Syntax Notation 1 (ASN.1, [ISO8824]. The first part of the future MHEG standard contains a formal specification of all data structures in ASN.1, the so called MHEG classes. The semantics of these MHEG classes define the functionality and requirements for an MHEG run-time environment.

MHEG Objects
An interactive multimedia presentation is composed by creation of instances of these classes. In the terminology of MHEG these instances are MHEG objects. As shown in Figure 2 we assume that MHEG objects are created on a sophisticated authoring workstation, interchanged to the end user by using several kinds of interchange media, and executed in a presentation terminal having scarce resources. In the presentation terminal an MHEG engine is responsible for the interpretation of MHEG objects to reconstruct the structure of the interactive multimedia presentation.

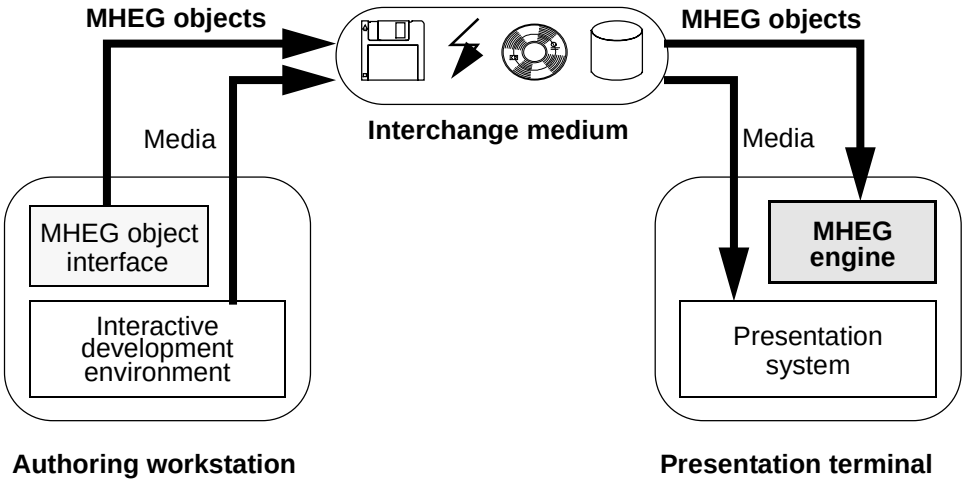

Figure 2: Life Cycle of MHEG objects

In order to interchange data from an authoring workstation to a presentation terminal, a transfer syntax is necessary. For each MHEG object in the abstract syntax, a corresponding bit encoding is defined in the transfer

syntax. The authoring workstation encodes its local data elements in the transfer syntax and the presentation terminal decodes them into its own local syntax. The ISO standard allows the two communication partners to bilaterally negotiate their own transfer syntax. But there is also an internationally standardized transfer syntax called Basic Encoding Rules for ASN.1 [ISO8825] which is preferred in MHEG. Depending on the kind of application, authoring workstation and presentation terminal have an encoder/decoder for these Basic Encoding Rules.

Run-time Objects

In order to use the data of MHEG objects in several instances during playback an author is able to create run-time objects (rt-objects) in the composition. For example multiple views on the same content object can be defined , where a content object is a reference to the real data. Rt-objects are not interchanged between the communicating systems; they are only available in the scope of the MHEG engine.

2.2 An Example for an Interactive Multimedia Presentation

Before we describe the MHEG objects in detail, we present an example of a multimedia presentation. A multimedia kiosk system [HoHe94] for retrieving video material is installed in front of a tourist information center. Figure 3 shows the time diagram for a session playing out the different media. The presentation starts with playing some music and showing a textual logo to catch the attention of pedestrians passing by. After several seconds, at a predefined position in the audio sequence, the text vanishes. When a voice recorded as a second audio sequence starts to speak, a graphic is shown on the screen as long as the voice is talking. The graphic has as a touch-sensitive area on the screen. In the time diagram a user interrupts the music by pushing the selectable area. A selection menu composed out of multiple text items then appears, and the user can enter a number into a data entry field in order to select the desired continuation of the presentation. For each selction a short video clip is provided.

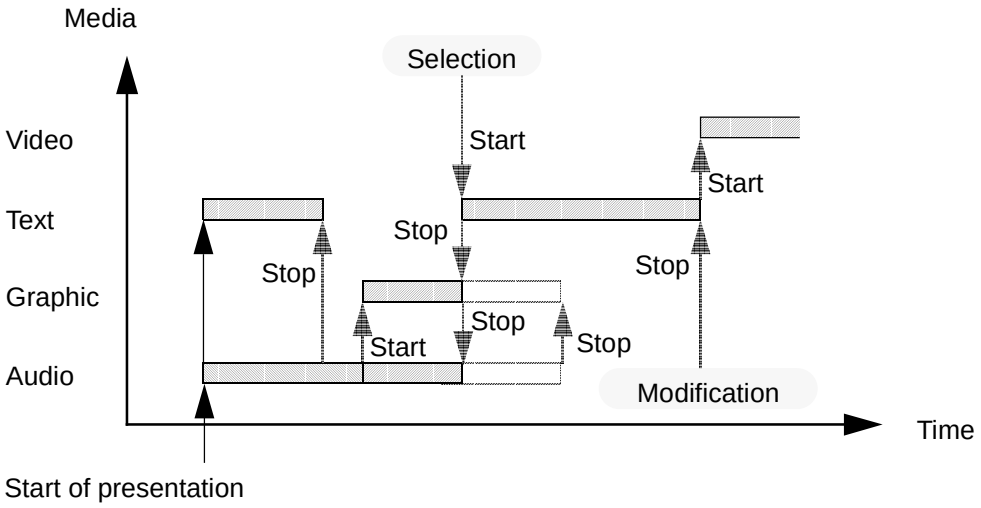

Figure 3: Time diagram of an Interactive Multimedia Presentation

2.3 Elements of the MHEG Model

Content Data
The presentation shown in Figure 3 consists of a sequence of pieces of information using different media. Each continuous piece of information is considered a single object. In our example the audio sequence, the graphic, the texts and the video sequence are *content objects* in the sense of MHEG.

Behavior
The term *behavior* means all activities concerning the actual presentation of objects on the man-machine interface. These activities have temporal and spatial aspects: actions such as Start, Stop and Set-Position control the behavior during playback. Temporal, spatial and conditional relationships between objects are controlled by links. Whenever the state of an object changes, this can be used to trigger activities in other objects, for example the end of the first audio sequence triggers the start of the second audio sequence.

User Interaction
It is important to understand that MHEG not only allows passive scripts with completely predetermined playback, but also allows user interaction. In the scenario of Figure 3 the running audio can be interrupted by user interaction. Two kinds of interactions are distinguished in MHEG: one is a simple *selection*, allowing the user to control future playback by choosing an alternative from a predefined set (e. g. push buttons). More complex interactions are called *modification*. An example is the user typing into a data entry field.

Composition
More complex MHEG objects can be composed from simple objects. In order to make MHEG objects interchangeable between systems, they can be packaged into containers. Just like in a hypertext or hypermedia document, these containers can be interconnected using hypertext links. Another major feature of *composite* objects is the synchronization of parts of the presentation.

Object Orientation
From our example it is obvious that the object-oriented approach is natural for use in MHEG. And MHEG is indeed object-oriented even if this method is used in a very weak sense for the specification of the MHEG classes. The only purpose of the class hierarchy in MHEG is abstraction with inheritance rules. Methods defining actions on the objects are not given in the standard. To describe the structure of MHEG classes, the possibility of importing, respectively exporting data types between ASN.1 modules is used.

We have now introduced the basic architectural principles of MHEG. In the next section we will present the most important classes of MHEG in detail by giving examples and hints for encoding the above presentation. We have to point out that this paper is written while the Draft International Standard [ISO13552] is sent out to the national bodies for balloting. Since we describe an evolving standard some of the details might be different in the final IS version of the document, but this does not effect the main concepts of MHEG in general.

3 Description of MHEG Objects

3.1 Common Attributes

Useful Definitions

The MHEG specification defines data structures which are often reused in multiple MHEG classes. To achieve type consistency these data structures are summarized in a module called *useful definitions*. For example generic identification mechanisms are used in the content, link, composite, and script class to reference other entities. There are three types of identification mechanisms: external, symbolic, and internal identification. The external identification is not defined by MHEG and therefore can be decoded for reference purposes without having to decode the referenced MHEG object. The symbolic identification may replace any other external or internal identification. For the internal identification the data structure *MHEG identifier* is used to address MHEG objects. It consists of two parts: The *application identifier* is a list of integer numbers; details are not defined in the document. The *object number* is another integer identifying objects uniquely within this application. Thus the MHEG identifier as a total uniquely identifies every single MHEG object world-wide. For the encoding of the kiosk scenario we use the internal identification mechanism to reference objects and file names to address content data externally.

MH-object class

The *MH-object class* is the root class of the MHEG class hierarchy. Its main purpose is to define two data structures common to all other MHEG classes; these data structures will be inherited by all lower-level classes. The data structure *class identifier* is used to identify the type of each class encoded by a predefined integer value. It is also possible to characterize every single MHEG object by a number of attributes. For example we use a database to store reusable MHEG objects, and it is reasonable to have a short description of each object also stored in the database. The attributes are used as search keys so that authors could find desired objects easily. All attributes of the description are optional, i.e. it is completely up to the application to define these data fields. An example is shown below.

```
Description {
   Name: "The object",
   Owner: "Thomas & Wolfgang",
   Version: "1.0",
   Date: "7/14/94",
   Copyright: "University of Mannheim",
   Comment: "no comment",
   Keywords: ("Test-object", ...)
   }
```

3.2 Content Data

The Content Class

The *content class* describes the real objects to be presented to the user. Conceptually every single content object is an atomic piece of information, containing the data necessary to present the object to the user. Every content object is of a particular medium type. This is similar to the content parts in MIME [Bore93]. The attribute *MHEG classification* defines the medium type. Either the digital data can be contained in the object itself, or the object contains a unique reference to the data stream. The first case is called included data. All included data will be run through the encoder/decoder for transfer; it is subject to the usual encoding/decoding

rules, similar to all other pieces of data exchanged between two MHEG entities. This is obviously only feasible for small amounts of data, such as text for subtitles, window titles, menu lists, labels on buttons, etc. The alternative is to send the data contents as *referenced data*. This implies that a unique reference is encoded in the object, and the digital data will be retrieved at run-time by the MHEG engine of the play-out site. The referencing mechanism is often also appropriate when the same large object is referenced in different multimedia presentations. This allows multiple presentations without actually copying the data.

For both included data and referenced data, MHEG provides a medium-dependent hook. The hook describes the encoding scheme used (*encoding identification*) and possibly parameters (*encoding description*). The hook allows the MHEG engine at the play-out site to initialize the appropriate presentation component at run-time. The standard document contains a detailed list of supported encoding formats. Examples are ISO 646 and ISO 6429 for text, ISO 8632 (CGM) for graphics, JPEG for still images, various PCM encodings for audio, and MPEG-Video and H.261 for video. For a complete list of supported formats, the reader is referred to the annex D of the MHEG document.

Virtual Coordinates

In addition to the encoding scheme, information on the *original size* of the object and the *original duration* of the play-out can be stored with the object. The measures for these are taken from the so-called generic space. It defines virtual coordinates on the x, y and z axes. Each axis goes from -32,768 to +32,767. At play-out time, the MHEG engine computes the physical coordinates of a visual MHEG object from the virtual coordinates in the object definition.

A virtual time coordinate is also defined. The interval goes from 0 to infinity and the unit is a millisecond. The temporal behavior of the play-out is created by mapping the definition of the object from the virtual time axis to the real-time requirements determined by the end user.

In the following example we present two content objects for the kiosk scenario of Figure 3. Object 1 includes the data for the text, and object 2 contains a referenced MPEG video clip. Following the CD specification the string “WELCOME” has to be encoded in a sequence of bytes, and the original size of the video is 256 x 240 pixels and the original duration is 15 minutes. The remaining content parts are encoded similarly.

```
Content-Class {
   MHEG-Identifier.Object-Number: 1,
   Classification: Text,
   Content-Hook {
      Encoding-Identification: ASCII},
   Content-Data.Data-Inclusion: "WELCOME"
}

Content-Class {
   MHEG-Identifier.Object-Number: 2,
   Classification: Video,
   Content-Hook {
      Encoding-Identification: ISO-11172-MPEG-Video,
      Encoding-Description: video rate in Kbps},
   Content-Data.Data-Reference.System-Identifier: "C:\video\test.mpg",
   Original-Perception {
      Original-Size : 256 pt, 240 pt, null,
      Original-Duration: 15 min}
}
```

Virtual Views
The purpose of the virtual coordinates described above is to avoid device dependencies in the description of multimedia objects, such as the number of pixels in the target window, or the audio sampling rate. In addition MHEG allows one to actually transform objects during play-out. An example would be to modify the volume of an audio sequence or to do clipping or zooming of a graphical object. The manner in which an MHEG object can be manipulated at play-out time is determined by parameters in the encoding (see Action class). Since MHEG is final form, it is not possible to modify the data itself (for example the color of the text can not be changed).

Let us assume that we want to present “WELCOME” multiple times on the kiosk screen and allow the end user to take a look from different perspectives. The text is now encoded as a two-dimensional graphical object in MHEG, allowing the modification of different parameters. These modifications are modelled as the invocation of a method on the object. In this way, different virtual views called *rt-content-objects* can be shown at presentation time. These virtual views are identified by unique numbers when the content object is originally composed, and contain the actual parameters at run-time when the object is played out.

Multiplexed Streams
Multimedia presentations may require access to individual streams in interleaved audio/video sequences like MPEG. Therefore a sub-class is derived from the content class called *multiplexed content class*. It contains or refers to the data with a description for each multiplexed stream. Single streams are accesible by a stream identifier encoded as an integer, for example to turn single streams on and off during playback. This class also allows dynamic multiplexing of multiple streams. This is a major requirement (e.g. lip synchronization) to interface inter-stream-synchronization mechanisms supported in many multimedia systems. Streams stored in individual content objects are grouped by the *set-multiplex* action into a single multiplexed content object.

3.3 Behavior

The Action Class
The action class is used to determine the behavior of basic MHEG objects. In object-oriented terminology, an action object is actually a message sent to an MHEG object or virtual view. This message invokes the corresponding method in the object, i.e. the execution of code within the body of the object. The result of the processing elementary actions is called MHEG effect. The definition of an action object is independent of the class on which it is invoked. It just represents the set of all publicly accessible MHEG objects. Implementation details are hidden from the user.

Not every action can be used on every object type. The MHEG specification contains a detailed list defining the actions allowed for each of the MHEG object types and also for virtual views. The property of polymorphism in object-oriented systems is used to allow the same action to execute on objects of different types; for example the play-out of an object is invoked with the RUN action no matter what the medium is.

States and Transitions
Some of the actions trigger a state transition in an MHEG object, which is relevant to other objects. Coming back to our scenario, reaching a certain position in the first audio triggered the appearance of the graphic, and the end of the second audio removes the graphic from the screen. In order to describe these relationships, it is

important to register the state of the virtual view of the audio. For this purpose the MHEG specification contains finite state machines for important actions of this type. Most of them are quite simple because of the small number of states. The *preparation status* represents the availability of an MHEG object. At initialization time the object is in state NOT READY. With the action PREPARE, referenced data is located, and the play-out component necessary for the presentation of the object is initialized. Then the object changes into state READY, and the content is presented on the output device using a virtual view. The action DESTROY frees all the resources allocated by the object. During processing of the MHEG effect for both actions the presentation status evaluates to the intermediate state PROCESSING. Whether the object is actually output to the user depends on the *presentation status* of the corresponding virtual view. This status distinguishes between a RUNNING, PROCESSING, and NOT-RUNNING state. *Timestones* allow the definition of triggers at predefined positions of a continuous stream.

The examples given above are atomic operations on an MHEG object or rt--object indicated by a *target parameter.* Composite actions can also be defined within an action object by composing atomic actions in a tree structure of nested actions. The target object is propagated from the top to the buttom of such a tree, as long as no other target is specified. A *synchro indicator* (serial, parallel) is introduced to allow a complex concurrent behavior in a presentation. If this parameter is omitted, then the atomic actions within a *synchronized actions list* are executed sequentially by default. The distinction between a parallel and serial execution has important consequences for an MHEG engine: the design and implementation of the interpretation process in the MHEG engine has to guarantee the concurrency of nested actions.

Below we show an example with two nested actions to start and stop the selected video in the kiosk scenario. The action object is sent to the target object 11 (the virtual view on the video encoded in content object 2 from our previous example). The first synchronized action contains three elementary actions: one to set the position, one to set the size of the video window, and finally one to show the video. The second synchronized action stops the video after twenty seconds. The synchronized actions are processed in parallel, whereas the elementary actions within each synchronized action are executed in serial. Similar action objects are needed for the start of the presentation, to remove the text, to present the graphic and for the user interaction etc.

```
Action-Class {
   MHEG-Identifier.Object-Number: 3,
   Synchro-Indicator: parallel,
   Target-Parameter.Generic-Reference: 11
   Synchronized-Actions (
      Action {
         Synchro-Indicator: serial,
         Synchronized-Actions (
            Rt-Object-Action-Attachment-Point: 250, 175,
            Rt-Object-Action.Set-Size: 140, 100,
            Rt-Object-Action.Run ) },
      Action {
         Synchro-Indicator: serial,
         Synchronized-Actions (
            General-Action.Delay: 20 sec,
            Rt-Object-Action.Stop ) } )
}
```

The Link Class

As discussed so far, there is no logical link between an action object and a content object or a virtual view. It is the purpose of the link class to define this logical relationship. A link class specifies under what conditions

actions are sent to other objects. At execution time each link instance corresponds to an event. When the event occurs the link is activated, and the action is sent to the targeted objects.

Typical multimedia presentations present several contents in parallel. In procedural script languages the author can often describe the play-out in the form of sequences of commands; in other words, the author has the responsibility for the synchronization of all the parallelism in his presentation, and he only has a sequential language to do that. For long and complex multimedia presentations, this is a very demanding and error-prone task if done by hand. The fact that MHEG links are triggered by events within the system allows the multimedia author to define logical relationships of causal or temporal nature, in the small. Simple “if... then...” rules can be used to describe these relationships. The synchronization is done by the execution environment at play-out time. It thus becomes much easier for the author to describe a complex multimedia presentation.

A link object always defines a relationship between exactly one source object and one or many target objects. The source objects and the target objects can be either MHEG objects or virtual views. The execution of a link object is triggered by the *trigger condition*. This condition is expressed by a state change in the source objects. As soon as the condition is fulfilled, the action objects listed within the link object body are sent to the set of target objects encoded in the action object. Although the standard gives no details on implementation, it can be assumed that the condition for the execution of action objects will be checked at run-time whenever the state of the source object changes.

Not all relationships can be expressed by conditions based on exactly one source object. Let us assume that an image is to be shown after both an audio sequence and a parallel video sequence are terminated. In this case the trigger condition is a conjunction of two conditions bound to different source objects. Such conditions can be added to the firing of a link using the *constraint condition* statement in MHEG. They express a required contextual state at the moment when the trigger condition is satisfied. If it is unclear which of the two source objects terminates first, the only possibility is to define both sequences of events separately, i.e. audio terminates first or video terminates first. This leads to the somewhat unnatural definition of two different trigger conditions in the same link object.

The trigger and constraint condition both evaluate boolean values. It is possible to set-up a logical combination of conditions using the logical operators: AND, OR, XOR, NAND, NOR, and NXOR. The representation of such a combination has the form of a logical tree, where the leaves are conditions and the logical operators are attached to the nodes.

In our ASN.1 example for a link object, we use the condition “When the presentation status in source object 10 (rt-audio) changes from NOT-MODIFIED to MODIFIED, then send action object 3.” Such link objects have to be encoded for each interaction (denoted by arrows in Figure 3) between objects in a presentation.

```
Link-Class {
   MHEG-Identifier.Object-Number: 4,
   Link-Condition {
      Source-Value.Get-Mofification-Status.Object-Number: 10,
      Previous-Condition.Evaluated-Condition {
         Comparison-Operator: equal,
         Compariso-Value.Modification-Status-Value: not-modified} }
      Current-Condition.Evaluated-Condition {
         Comparison-Operator: equal,
         Comparison-Value.Modification-Status-Value: modified} }
```

```
    Link-Effect.Action.Object-Number: 3
}
```

It is not only possible to trigger certain status fields; it is also allowed to trigger parameters of rt-objects. For example a link can be defined dependent on the adjustment of the volume parameter attached to an audio object. Furthermore a macro facility provides an author to reuse complex action objects. A macro is comparable to the well known `#define` clause in C preprocessors. Parameters of elementary actions can be changed at run-time before the action is send to the target object. This is a generalization of the primitive link model, it enables the composition of more effective presentations.

3.4 User Interaction

With the content class, the action class and the link class introduced so far, we can compose complex multimedia presentations, with parallel play-out of content objects. However, they will execute after start up without the possibility of user interaction. The *selection* is the construct allowing simple user interaction out of a predefined set of alternatives whereas the *modification* is used for more complex data input.

Selection
The discussion of the link class has shown how the play-out of a presentation can be influenced by events occurring at a predefined point in time by the author. With the selection behavior, it is now possible to create such events at run-time by the user. The definition of such events is done by assigning for each of the possible interaction alternatives a corresponding internal event. When the user interacts with the system, e.g. presses a button, the corresponding *selection status* changes from NOT-SELECTED to SELECTED and the assigned value is set to the attribute *selection mode*. A change of the selection status can be used to trigger a link object and the selection mode to evaluate the constraint condition. The selection behavior is associated with a rt-content object, for example a graphic on the screen. Hierarchical selections such as pull down menus can also be defined.

Modification
The second and more general form of interaction in MHEG is the modification behavior. Its purpose is the entry and manipulation of data. As opposed to the selection behavior, the modification behavior has no predetermined set of alternatives, but the result of the interaction is internally represented as a content object. The actual status of the content object is registered as the state of the object before, during and after modification. A content object of type numeric or string is used to store the user's data input. The value contained in the *content data* part of the content object, is used as the initial value to be modified. It is thus possible, similar to the selection, to include a modification status into the condition of a link object and thus influence the presentation based on the user's data input. Values stored in the content object may be retrieved as parameters for elementary actions.

Styles
Selection and modification are described as a behavior of a rt-content-object. The style of these kinds of user interaction is defined by the interaction behavior. MHEG contains five predefined styles: button, slider, entry field, menue, and scrolling list. For example the action *set-button-style* will attach the selection status to the graphic object to behave as a button and *set-entryfield-style* the modification status to the text to perform as an entry field. An author is able to define the look and feel of the user interaction or to use primitives provided by

the graphical user interface of the executing MHEG engine.

The following actions encoded the selection of the rt-content-object 50 - a virtual view of the graphic - to interrupt the playback of the 'welcome animation' in the kiosk scenario. We do not encode an alternative representation of the button in the case it is selected.

```
Set-Button-Style {
   Targets: (50),
   Initial-State: selectable-not-selected
}

Set-Selectability {
   Perceptible-Target-Reference: 50,
   Min-Number-Selections: 1,
   Max-Number-Selections: 1
}
```

In our example for a modification behavior, the rt-content-object 100 references a content object to store numerical values, representing the number of the video which the user wants to see.

```
Set-Entryfield-Style {
   Targets: (100)
}
```

3.5 Composition

The encoding described so far sets up a collection of individual objects defining specific aspects of the kiosk scenario. Although MHEG does not prevent to interchange these objects separately a data structure is needed to compose the multimedia presentation.

The Composite Class

The skeleton of a presentation is given by the composite class. In a first step objects belonging to a certain part of the presentation have to be grouped together. The recursive definition of the composite class allows that composite objects can be part of other composite objects.

From a logical point of view a composite object can be understood as kind of container. Comparable to the content class MHEG distinguishes between included or referenced objects in the composite class. External references are helpful to split up a complex presentation into smaller pieces to reduce the amount of memory needed in the end system, and to support real-time requirements by partial object retrieval. For example references between composite objects are used to set up a hypertext-like structure. The access to referenced objects has to be established by appropriate services in the execution environment.

The second task of the composite class is to associated the objects in a way that they can be executed to reconstruct the encoded presentation. This is performed by four default link objects the *availability-start-up/close-down* and the *rt-availability-start-up/close-down* which have implicit defined trigger conditions. For example the availability-start-up "whenever the composite object is ready/not-ready then..." we used to prepare all referenced objects in the composite object. The rt-availability-star-up "when an rt-composite is created from this composite" we used to invoke the action objects needed to start the presentation.

The following rudimentary encoding refers to the initial composite object of the scenario. All link and action objects are included or referenced in the attribute *specific behavior.* The attribute *composition* includes or references the content objects needed in this t part of the presentation.

```
Component-Class {
  MHEG-Identifier.Object-Number: 6
  Composition-Behavior {
    Predefined-Behavior {
      Availability-Start-up: ...,
      Availability-Close-Down: ...,
      Rt-Availability-Start-up: ...,
      Rt-Availability-Close-Down: ...},
    Specific-Behavior {
      Actions (list of actions objects),
      Links (list of link objects) }},
  Composition {
    Number-of-Elements: ...,
    Elements (list of content and composite objects) }
}
```

Further Classes

The MHEG standard defines three more classes: the container class, the descriptor class and the script class. The container class provides a package of multiple objects in order to interchange them as a whole set, but without information how to reconstruct the presentation. The descriptor class is used to encode information about the objects of a presentation, for example which media representations are used for the content objects. Furthermore a descriptor object might contain quality of service information to support real-time interchange of MHEG objects. This can be used by an MHEG engine to evaluate the requirements of the presentation with respect to the available resources of the run-time platform. The script class is an interface to external functions or programs. A typical application of the script class can be a database query. The query input will be encoded by suitable MHEG objects whereas a script object is used to invoke the query executed in the script environment and to transmit data between the MHEG engine and the script environment. For all the details of these two classes the interested reader is referred to the MHEG document.

4 MHEG Implementation Issues

4.1 Architecture of the MHEG Run-time Environment

Whereas the standard describes the MHEG object structure and transfer encoding in great detail, very little is said about the implementation of an MHEG engine. Early experience with MHEG implementation environments is currently gained in prototype systems. The work presented in this section is largely based on a joint project between IBM's European Networking Center in Heidelberg and the University of Mannheim. Figure 4 presents the prototype developed for a networked multimedia kiosk environment. This environment consists of presentation terminals running the MHEG engine. These presentation terminals are connected to a remote database server to retrieve MHE- encoded interactive multimedia presentations. An interactive Editor is provided on an authoring workstation for the composition of the MHEG objects. The implementation of the prototype is based on the CD and is now available under IBM AIX for RS/6000. As a port to personal computers is envisaged, the MHEG editor and the MHEG database are also available for IBM OS/2 2.1.

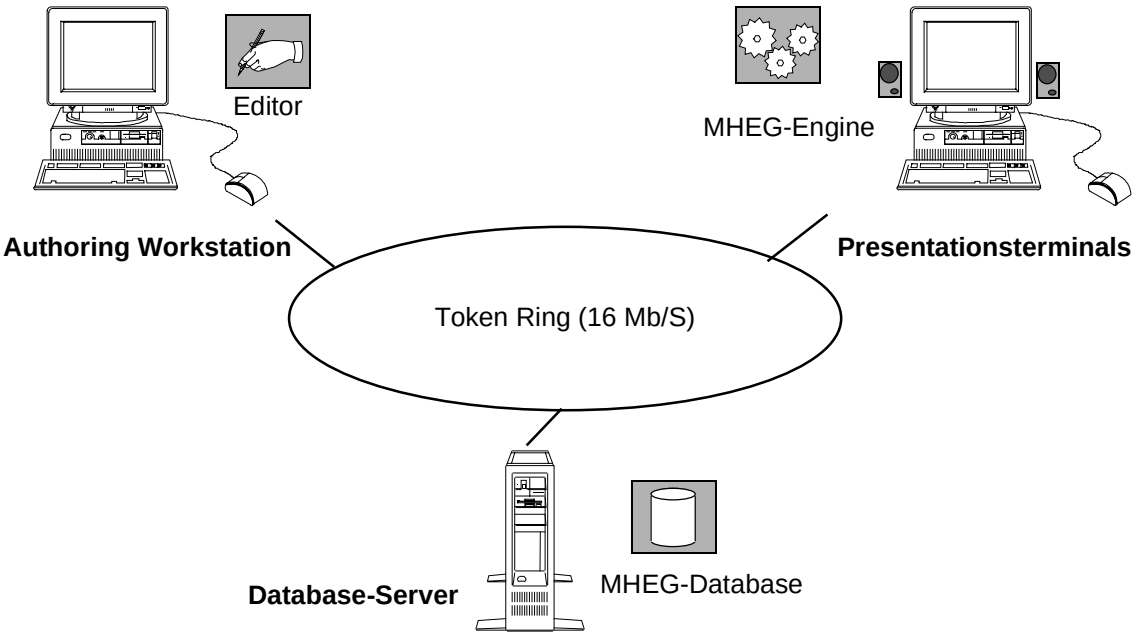

Figure 4: MHEG based networked kiosk system

Recently [Bert94] in France and [SHKS] in Korea have presented two MHEG run-time environments similar to use. Both prototypes were implemented as distributed retrieval systems for Microsoft Windows 3.1. The Korean implementors report problems with the non-preemptive scheduler of Microsoft Windows. Due to the concurrency requirements of MHEG's event-driven execution model and the real-time constraints of multimedia data we recommend to implement an MHEG engine in an operating system supporting preemptive multitasking.

Authoring an MHEG Presentation

The MHEG presentation author starts with a set of existing basic objects such as texts, images, audio sequences and video streams created and manipulated by appropriate tools installed on the authoring workstation to draw pictures, cut video, record audio etc. Each of them must be in one of the standardized formats supported by the MHEG run-time environment. The MHEG presentation is now composed out of these content objects.

A number of authoring tools is already available in the market (e.g. Macromind Director). Most of them have their own script language (e.g. Lingo, IBM's AVA/2, Script/X) supporting the author by application-oriented semantics. It is also possible to use editors from hypertext or hypermedia systems, such as Hypercard, or text processors. All of these authoring tools have the same purpose, and the most promising method for producing MHEG presentations would be to enhance these tools with an MHEG output. Another possibility would be a converter translating between the proprietary format of an existing authoring tool and the MHEG object format described above (for example from HyTime to MHEG [Mark91]). With respect to script languages there will be hard problems converting to MHEG, because the link-model is too primitive.

As mentioned above the standardized representation of MHEG objects is in ASN.1. This language is at a very

low level of detail, and we do not recommended to author sophisticated multimedia presentations directly at this level. This would be comparable to programming complex systems in assembly language. Nevertheless we implemented a simple interactive editor to create, modify and delete MHEG-objects at this level.

The individual MHEG objects are stored as a binary data stream (transfer syntax) in a central SQL database server. The data structures *description* and *MHEG-Identifier* were extracted from each MHEG object and used as attributes and keys in the database schema.

Distribution of MHEG Objects
Once an presentation is completely encoded, it can be transmitted through all forms of information exchange. Typical examples would be CD-ROM distribution or transfer over networks. The exchange of a standardized MHEG object boils down to the exchange of a standardized binary file between heterogeneous systems, and any kind of file transfer mechanism can be used. A comparable approach is under development by Kaleida. The Script/X authoring environment also supports a lower level interface - the *Script/X Byte Code* [Kale94].

In our architecture MHEG objects are interchanged between the editor in the authoring workstation and the database server as well as between the database server and the presentation terminal over the network. Communication was enabled by a client-server protocol called MHEG request/response protocol (MRP) over TCP.

4.2 Playback at the Presentation Terminal

On the end user's computer system the MHEG presentation is shown using an *MHEG engine*. The MHEG engine can decode all of the MHEG objects, can handle MHEG events and links, and can execute the interpreters for all the basic content object types of MHEG. Possible user interactions were programmed by the author using the selection and modification classes, and are only possible at points preplanned by the author. Figure 5 shows the flow of MHEG objects through the components of an MHEG engine.

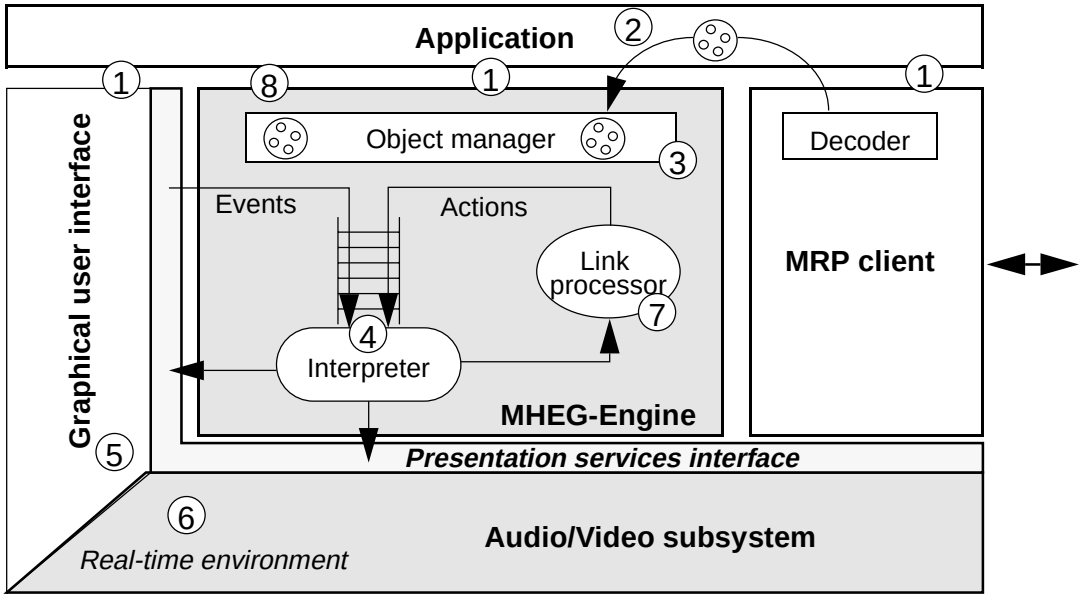

Figure 5: Components of an MHEG Engine

Let us now consider the steps taken during playback of the MHEG-encoded kiosk presentation:

1. In the first step the application initializes the MHEG engine, the MRP client and also the presentation services. The latter typically comprise the local audio and video drivers and the window-based user interface. The initialization also prepares the first object of the MHEG presentation for execution; this is normally the first composite object found on a local file or retrieved over the network. After the start of the presentation, additional objects can be retrieved.

2. The MRP client provides network transparent access to all MHEG objects. Before they can be interpreted, they have to decoded from the transfer syntax (i.e. Basic Encoding ules for ASN.1) into the local syntax. After the transfer is completed the applcation starts the presentation by sending this root object to the MHEG engine.

3. The object manager is a central resource storing all MHEG objects loaded into the engine during the execution cycle. Before they can be interpreted, they have to be decoded from the transfer syntax (i.e. Basic Encoding Rules for ASN.1) into the local syntax. If an MHEG object contains a reference to another object, the object manager also requests thetransfer of these objects.

4. The interpreter is the main component of the MHEG engine. It is event-driven, with all the waiting events maintained in a message queue. Basically there are two types of events, those coming from the local presentation services and those activated by the link processor. Both types of events relate to a particular object or a virtual view and can be evaluated and executed by the interpreter accordingly. The first object of a presentation is usually started using the action PREPARE on the active composite object.

5. During interpretation the MHEG interpreter calls the presentation services. It is the task of the presentation services to implement th rt-content-objects. For example the action RUN on a text object will show its data on the user's screen. Discrete media (text, graphic, etc.) and primitives such as sliders, buttons, and entry fileds provided by the graphical user interface are executed in the non-real-time environment of the presentation services.

6. For the playback of continuous media, it is assumed that the computer contains a special audio-video subsystem with corresponding drivers. This subsystem provides a real-time environment where processes are controlled by appropriate scheduling policies to guarantee the synchronized playback of continuous media. Typically these drivers support the basic contents types defined in the MHEG standard, such as MPEG video or PCM-encoded audio, directly. Often these media streams are not contained in the MHEG objects but referenced. In this case the digitized audio or video data have to be retrieved from a multimedia file system or from remote via a multimedia transport system.

7. When an incoming message is processed, the status in the various status fields of the interpreter can be changed. Examples are the timestone, presentation, selection and modification status. In these cases the link processor is activated. It checks all the active link objects for which the modified MHEG is defined as a source object. If both the trigger condition and possible additional conditions are fulfilled then the actions defined in the link object are sent to the target object by enqueuing them into the central message queue.

8. After all composite objects of a presentation have been explicitly, the MHEG engine frees all local

resources and is ready to start a new presentation.

It should be noted that the future MHEG standard does not describe the components of the MHEG engine or the details of the interface between the MHEG engine and a local application. This is considered to be an implementation issue. For specific multimedia applications such interfaces are currently being defined (see for example ITU Recommendation T.170 [ITUT170]).

5 Conclusions and Outlook

Until today, research and development has concentrated on coding and compression schemes for individual media types, and on high level languages for document structures. But the exchange and playback of multimedia documents requires not only standardized formats for media contents, but also for the structure of a complex, interactive multimedia presentation. This is the purpose of the fothcoming MHEG standard.It will allow the distribution of complex multimedia information in final form and the play-out on heterogeneous systems. this is an enabling technology for a large number of new applications.

The standardization process is now reaching DIS level in the ISO committee, and MHEG prototypes are under development. It is very important to use feedback from early implementations to improve the MHEG specification. For example, out of our own work, the following contributions were made: Redesign of the user interaction to allow a more flexible encoding of the look and feel of kiosk application, introduction of *get-actions* to access the values stored in alphanumeric content object, logical combination of conditions represented in form of a tree structure, and support of a group concept to synchronize multimple continuous streams stored in individual content objects.

It is hoped that the exchange of complex multimedia information between heterogeneous systems will soon be a reality. Actually a DIS like version of the presentation terminal for OS/2 is developed in the project Globally Accessible Services, a german project sponsored by DeTeBerkom.

Acknowledgements

The implementation project described in this paper was done at IBM’s European Networking Center in Heidelberg. We would like to thank IBM for the financial support and our colleagus at ENC for the excellent cooperation. Special thanks go to Ralf Steinmetz for his helpful comments on an earlier version of the paper, and to Guido Grassel for coding many parts of the prototype.

References

[Bert94] Christian Bertin, "Eurescom IMS1 Projects (Integrated Multimedia Services at about 1 Mbit/s)", Proc. of the 2nd International Workshop on Multimedia: Advanced Teleservices and High-Speed Communication Architectures, Heidelberg, Sep. 26-28, 1994, R. Steinmetz (Ed.), Springer LNCS vol.868, 1994, pp.53-66.

[Bore93] N.S. Borenstein, "MIME - A Portable and Robust Multimedia Format for Internet Mail", ACM Multimedia Systems (1993) 1, published by Springer, pp.29-36.

[BuZe93] M.C. Buchanan, P.T. Zellweger, "Automatic Temporal Layout Mechanisms", Proceedings of the 1st ACM International Conference on Multimedia, Anaheim (CA), USA, Aug. 1-6, 1993, pp.341-350.

[HoHe94] W. Holfelder, D. Hehmann, "A Networked Multimedia Retrival Management System for Distributed Multimedia Kiosk System Support", Proc. of the 1st IEEE Conference on Multimedia Computing and Systems, Boston (MA), USA, May 14-19, 1994, pp.132-143.

[ISO8824] ISO/IEC IS 8824:1987, "Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)".

[ISO8825] ISO/IEC IS 8825:1987, "Information Processing Systems - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) with DAD1".

[ISO10918] ISO/IEC IS 10918:1992, "Information Technology - Digital Compression and Coding of Continuous-Tone Still Images (JPEG)".

[ISO11172] 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)".

[ISO13552] ISO/IEC CD 13552-1:1993, "Information Technology - Coded Representation of Multimedia and Hypermedia Information Objects (MHEG) - Part 1: Base Natotion (ASN.1)", 15. October 1994.

[ITUT170] ITU-T Draft Recommondation T.170, "Audiovisual Interactive (AVI) Systems - General Introduction, Principles, Concepts and Models", Second Revision, Geneva, CH, 16-25 Nov. 1993.

[Kale94] Kaleida Labs Inc., "Kaleida, Script/X and the Multimedia Market", Kaleida White Paper, Revision 1, 1945 Charlston Road, Mountain View, CA 94043, USA, 1994.

[KrCo92] F. Kretz, F. Colaitis, "Standardizing Hypermedia Information Objects", IEEE Communications Magazine, vol.1, no.5, May, 1992, pp. 60-70.

[Mark91] B. Markey, "Emerging Hypermedia Standards: Hypermedia Marketplace Prepares for HyTime and MHEG, (U.S.) Assistant Secretary of Defense (Production and Logistics), Washington, DC, PB92-120328, 1991.

[Pric93] R. Price, "MHEG: An Introduction to the Future International Standard for Hypermedia Interchange", Proc. of the 1st ACM International Conference on Multimedia, Anaheim (CA), USA, Aug. 1-6 1993, pp.121-128.

[SHKH94] Jong-Jin Sung, Mi-Young Huh, Hyoung-Jun Kim, Jin-Ho Hahm, "Hypermedia Information Retrieval System Using MHEG Coded Representation in a Networked Environment", Proc. of the 2nd International Workshop on Multimedia: Advanced Teleservices and High-Speed Communication Architectures, Heidelberg, Sep. 26-28, 1994, R. Steinmetz (Ed.), Springer LNCS vol.868, 1994, pp.67-77.

Acronyms

ASN.1	Abstract Syntax Notation Number One
AVA/2	Audio Visual Authoring/2
BER	Basic Encoding Rules
CD	Committee Draft
DIS	Draft International Standard
ISO	International Organization for Standardization
ITU	International Telecommunication Union
JPEG	Joint Photographic Experts Group
MHEG	Multimedia Hypermedia Experts Group
MIME	Multipurpose Internet Mail Extensions
MRP	MHEG Request/response Protocol
MPEG	Motion Picture Experts Group
SQL	Structured Query Language
TCP	Transmission Control Protocol