MHEG Explained

Thomas Meyer-Boudnik and Wolfgang Effelsberg
University of Mannheim

The upcoming MHEG standard, under development by ISO's Multimedia Hypermedia Experts Group, defines a system-independent encoding of the structure information used for storing, exchanging, and executing multimedia presentations. This article describes the building blocks of an MHEG-encoded presentation through a running example. Our experience involved using MHEG in a networked multimedia kiosk, for which we present a complete MHEG runtime environment.

The multimedia market is characterized by numerous manufacturers, each using its own technology for digital audio and video, script languages, encoders and decoders, communication protocols, operating system extensions, and so on. There is currently no standardized way to compose a complex, interactive multimedia presentation, store it on a medium, send it over a network, and play it out on somebody else's computer. The situation echoes networking in the late 1970s, when every manufacturer had its own networking architecture and heterogeneous networks of computers could hardly interoperate. In the networking field the definition of protocol standards created a tremendous potential for interchanging information between all kinds of computers world-wide. We expect that creating open standards for multimedia presentations will have a similar impact.

Information providers today have a strong interest in creating multimedia presentations, such as books enhanced by audio annotations or short video clips. Computer companies and major publishers created many joint ventures to get actively involved in this market. But it remains unclear in what document architecture an author should create a document for maximum portability. No single system has penetrated the world market and established a de facto standard. For example, a multimedia book would have to include several video formats, such as Compact Disk Interactive (CD-I), MPEG-1, and Apple's QuickTime, to reach a mass market. This is inconvenient and expensive for the producer.

Similarly, a typical multimedia end user must own several different systems to view all the available presentations; most important would be an IBM-compatible PC, an Apple Macintosh, and a Unix workstation. Those having only one system cannot access all the available information. As a result of these problems, both the multimedia authoring community and multimedia consumers have strong motives to define a universal standard and implement it on all major platforms.

Multimedia technology is now readily available on many computer systems. Storage and playback typically take place locally, without transmission over networks or exchange of data between heterogeneous systems. The problem is the limited use of international standards for representing multimedia content, in particular conditional links, and spatial and temporal relationships between such content elements in final form (multimedia script). The International Organization for Standardization (ISO) has addressed this problem in subcommittee ISO/IEC JTC1/SC29 (Coding of Audio, Picture, Multimedia, and Hypermedia Information). The subcommittee has three working groups: JBIG/JPEG, MPEG, and MHEG.

The JPEG standard defines compression algorithms for still images.[1] When applied to video, the compression algorithms are often called Motion JPEG. Unfortunately, the compression algorithms cannot take into account the temporal redundancy in a video stream, since JPEG compresses each frame in isolation. Therefore, ISO created the MPEG standard, which defines a compression and interchange format for motion pictures, including audio.[2] Both JPEG and MPEG are well known and widely used in multimedia systems.

The JPEG and MPEG standards both describe the content of information objects only. They cannot 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 outlined the scope of the standard.[3,4] In this article we present the basic ideas behind MHEG in more detail, starting with the MHEG model and moving into the classes and their relationships. A running example further clarifies each class in a slightly simplified ASN.1 notation. We also discuss how to create and transmit an MHEG presentation, and how to execute it in an MHEG runtime environment.

MHEG's data interchange model

Although an interchange format, MHEG is more

1070-986X/95/$4.00 © 1995 IEEE

than just a binary format. It also possesses features that suit real-time interchange in a networked environment. Before describing these features, we introduce the several levels of representation—classes, objects, and runtime 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." ISO standardized a notation called Abstract Syntax Notation 1 (ASN.1) to allow the definition of abstract syntaxes.[5] 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 runtime environment.

MHEG objects

An interactive multimedia presentation is composed by creating instances of MHEG classes. In the terminology of MHEG, these instances are *MHEG objects*. As shown in Figure 1, we assume that MHEG objects are created on a sophisticated authoring workstation, communicated to the end user through several kinds of interchange media, and executed in a presentation terminal having scarce resources. An MHEG engine in the presentation terminal is responsible for interpreting MHEG objects to reconstruct the structure of the interactive multimedia presentation.

Sending data from an authoring workstation to a presentation terminal requires a transfer syntax. Each MHEG object has a well-defined bit encoding 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. However, the internationally standardized transfer syntax called Basic Encoding Rules for ASN.1[6] is preferred in MHEG. BER encoders and decoders must be provided for authoring workstations and presentation terminals, depending on the kind of application.

Acronyms

ASN.1	Abstract Syntax Notation One
AVA/2	Audio Visual Authoring/2
BER	Basic Encoding Rules
CD	committee draft
DIS	Draft International Standard
ISO	International Standards Organization
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	Moving Pictures Experts Group
SQL	Structured Query Language

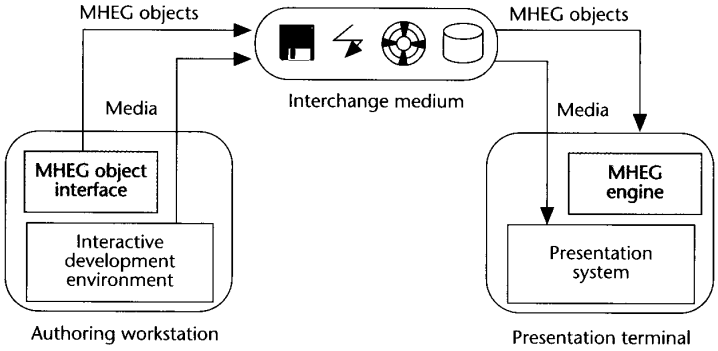

Figure 1. The life cycle of MHEG objects.

Runtime objects

To use the data of MHEG objects in several instances during playback, an author can create runtime objects (rt-objects) in the composition. For example, the author can define multiple views on the same *content object*, 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 MHEG engine.

Example presentation

Before describing the MHEG objects in detail, we present an example of a multimedia presentation. A multimedia kiosk[7] is installed in front of a tourist information center for public access. It features a touch screen for user interaction and speakers for audio output. Figure 2 shows the

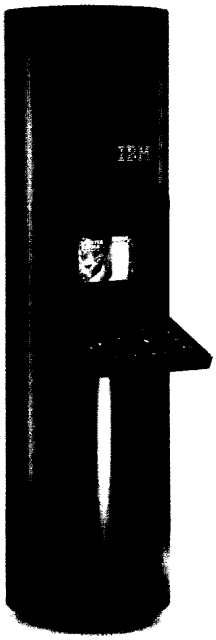

Figure 2. The kiosk used in our MHEG presentation terminal example throughout this article.

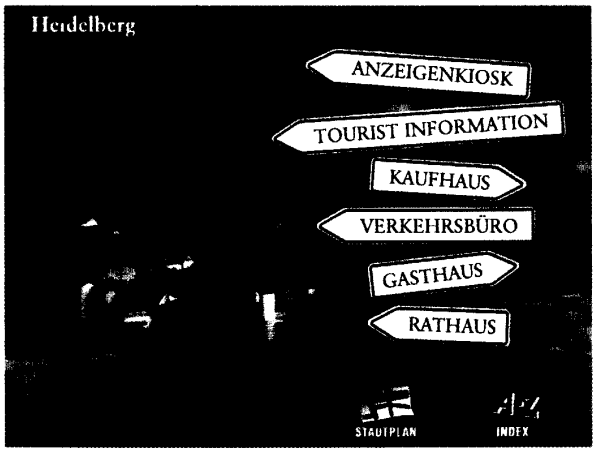

Figure 3. A screen shot of the initial screen that appears on the kiosk, showing pedestrians—the potential users—the options available to them.

kiosk; Figure 3 is a screen shot of the initial screen. We use the kiosk as our MHEG presentation terminal throughout this article. Figure 4 shows the time diagram for a session playing out the different media. The presentation starts by 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 appears on the screen as long as the voice continues. The graphic has a touch-sensitive area, as shown in Figure 5. In the time diagram a user interrupts the music by pressing the selectable area. A selection menu composed of multiple text items then appears. The user can enter a number into a data entry field to select a short video clip to be shown as the next segment of the presentation.

Elements of the MHEG model

Now we look at the MHEG model's elements in more detail. We must point out that we wrote this article while the Draft International Standard[8] was with the national bodies for balloting. Since we describe an evolving standard, some of the details might differ from the final ISO version of the document, but those differences will probably not affect the main concepts of MHEG.

Content data. The presentation shown in Figure 4 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 sequences are content objects.

Behavior. The term "behavior" means all activities concerning the actual presentation of objects on the human-machine interface. These activities have temporal and spatial aspects: actions such as Start, Stop, and Set-Position control behavior during playback. Links control temporal, spatial, and conditional relationships between objects. This allows an author to use a change in the state of an object 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. MHEG not only allows passive scripts with completely predetermined playback, it also allows user interaction. In the scenario of Figure 4, for example, user interaction can interrupt the running audio. MHEG distinguishes two kinds of interaction:

1. A simple *selection*, which allows the user to control future playback by choosing an alternative from a predefined set (for example, push buttons).

2. More complex interactions, called *modification*, such as the user typing into a data entry field.

Composition. More complex MHEG objects can be composed from simple objects. To make MHEG objects interchangeable between systems, you can package them into containers. Just like in a hypertext or hypermedia document, you can

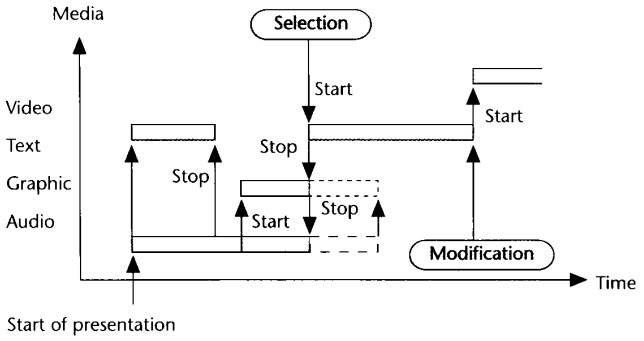

Figure 4. As this time diagram of our kiosk application shows, MHEG allows user interaction to control playback of a presentation.

interconnect these containers using hypertext links. Composite objects also allow synchronization of parts of the presentation.

Object orientation. Our example shows 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 specification of the MHEG classes. The only purpose of the class hierarchy in MHEG is to provide abstraction with inheritance rules. The standard does not give methods defining actions on the objects. To describe the structure of MHEG classes, the standard uses the possibility of importing and exporting data types between ASN.1 modules.

Description of MHEG objects

In describing the MHEG objects, we look more closely at common attributes, content data, behavior, user interaction, and composition. Remember, however, that MHEG is still in the draft stage and so is subject to change. Coding samples help demonstrate some of the concepts.

Common attributes

MHEG's common attributes provide a module called "useful definitions" to achieve type consistency of data structures, and the MH-object class, which is the root class of the MHEG class hierarchy.

Useful definitions. The MHEG specification defines data structures that are often reused in multiple MHEG classes. To achieve type consistency, these data structures are summarized in the module *useful definitions*. For example, generic identification mechanisms are used in the content, link, composite, and script to reference other entities.

There are three types of identification mechanism: external, symbolic, and internal. External identification is not defined by MHEG and therefore can be decoded for reference purposes without having to decode the referenced MHEG object. Symbolic identification may replace any other external or internal identification. For internal identification, the data structure *MHEG identifier* contains the address of an MHEG object. 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 within this application. Thus the MHEG identifier as a total uniquely identifies every single MHEG object.

For the encoding of the kiosk scenario, we used the MHEG identifier to reference objects internally and file names to address content data externally.

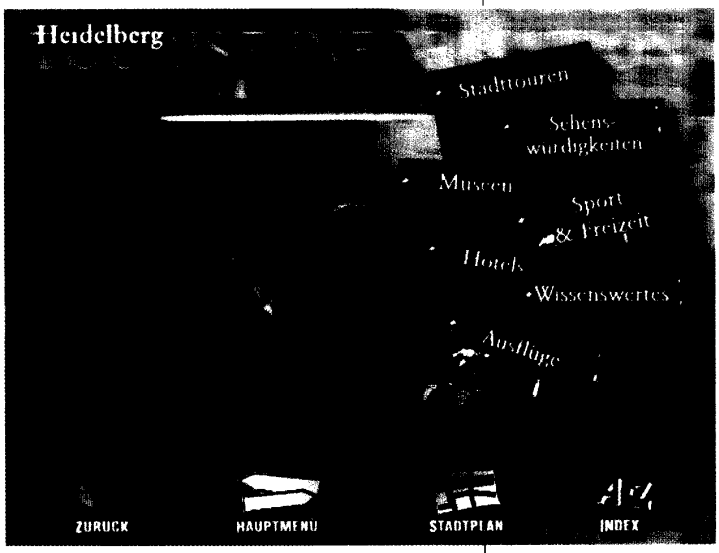

Figure 5. The graphic shown on the kiosk screen features a touch-sensitive area, which the user can press to call up different options.

The 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 and inherited by all lower level classes. The data structure *class identifier* identifies the type of each class encoded by a predefined integer value.

It is also possible to characterize each MHEG object by a number of attributes. For example, we use a database to store reusable MHEG objects; it seems reasonable to have a short description of each object also stored in the database. The attributes serve as search keys so that authors can find desired objects easily. All attributes of the description are optional—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", ...)
   }
```

Content data

To understand content data, we need to consider the content class, virtual coordinates, virtual views, and multiplexed streams. The two sample content objects introduced to demonstrate virtual coordinates refer back to the kiosk scenario of our running example.

The content class. The *content class* describes the real objects 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, similar to the content parts in MIME.[9] The attribute *MHEG classification* defines the medium type. The digital data either is contained in the object itself, or else the object contains a unique reference to the data stream.

The first case is called *included data*. All included data runs through the encoder/decoder for transfer, 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, and so forth. 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 runtime by the MHEG engine of the play-out site. The referencing mechanism is often also appropriate when the same large object is used in different multimedia presentations. This allows multiple presentations without actually copying the data.

MHEG provides a medium-dependent hook for both included data and referenced data. 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 runtime. MHEG supports a number of encoding formats, such as 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, consult annex D of the MHEG document.[8]

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 generic space, which 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.

The MHEG document also defines a virtual time coordinate, with an interval from 0 to infinity measured in millisecond units. 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 example shown in Figure 6 we present two content objects for the kiosk scenario of Figures 2 to 5. Object 1 includes the data for the text, and object 2 contains a referenced MPEG video clip. The string "WELCOME" is encoded in a sequence of bytes. The original size of the video is 256 × 240 pixels, and the original duration is 15 minutes. The remaining content parts are encoded similarly.

Virtual views. The purpose of the virtual coordinates 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 lets you actually transform objects during play-out. For example, you could modify the volume of an audio sequence, or do clipping or zooming of a graphical object. Parameters in the encoding

determine the manner in which an MHEG object can be manipulated at play-out time (see the action class, discussed in the section "Behavior"). Since MHEG is a final form representation, you cannot modify the data itself (for example, you cannot change the color of the text).

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 modification of different parameters. These modifications are modeled 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, identified by unique numbers when the content object is originally composed, contain the actual parameters at runtime when the object plays out.

Multiplexed streams. Multimedia presentations may require access to individual streams in interleaved audio/video sequences like MPEG. Therefore, a subclass called *multiplexed content class* is derived from the content class. It contains or refers to the data with a description for each multiplexed stream. A stream identifier encoded as an integer can be used to control single streams, for example, to turn audio on and off in an MPEG system stream.

This class also allows dynamic multiplexing of multiple streams. This is a major requirement for interstream synchronization mechanisms supported in many multimedia systems (consider lip synchronization). Streams stored in individual content objects are grouped by the action *Set-multiplex* into a single multiplexed content object.

Behavior

Various features in MHEG allow the description of behavior. We look at the action class, states and transitions, and the link class. In this section, the example based on the kiosk scenario presents an action object and a link object.

The action class. The action class determines 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, by executing code within the body of the object. The result of processing elementary actions is called the *MHEG effect*. Implementation details remain hidden from the user.

Object 1

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

Object 2

```
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}
}
```

Figure 6. Two content objects for the kiosk scenario of Figure 4. Object 1 includes the data for the text, and object 2 contains a referenced MPEG video clip.

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. Similar to other object-oriented systems, the property of polymorphism allows the same action to execute on objects of different types. For example, the play-out of an object is invoked with the action Run regardless of the medium.

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 removed the graphic from the screen. To describe these relationships, it is important to register the state of the virtual view of the audio. The MHEG specification contains finite-state machines for important actions of this type. Most are quite simple because of the small number of states.

For example, the preparation status represents the availability of an MHEG object. At initialization time the object is in the state "Not Ready." The action "Prepare" locates referenced data and initializes the play-out component necessary for presentation of the object. The object changes into the state "Ready" automatically, and the con-

tent is presented on the output device using a virtual view. The action "Destroy" frees all resources allocated by the object.

The examples given above are atomic operations on an MHEG object or rt-object indicated by a *target parameter*. You can also define composite actions 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 bottom of such a tree, as long as you specify no other target.

The draft standard defines a *synchro indicator* (serial or parallel) to allow complex concurrent behavior in a presentation. If you omit this parameter, then by default the atomic actions within a synchronized actions list execute sequentially. The distinction between parallel and serial execution has important consequences for an MHEG engine: The design and implementation of the interpretation process in the MHEG engine must guarantee the concurrency of nested actions.

```
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 ) } )
}
```

Figure 7. Two nested actions start and stop the selected video in the kiosk scenario.

The example in Figure 7 shows two nested actions to start and stop the selected video in the kiosk scenario. The action object is sent to target object 11 (the virtual view on the video encoded in content object 2 from Figure 6). 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 20 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, for the user interaction, and so forth.

The link class. As discussed so far, there is no logical link between an action object and a content object or a virtual view. The link class defines 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 offer several contents in parallel. In procedural script languages an author can often describe the play-out as sequences of commands. In other words, authors are responsible for synchronizing all the parallelism in their presentations, and they have only 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 lets multimedia authors define logical relationships of causal or temporal nature, in the small. Simple "if-then" rules can describe these relationships. The execution environment does the synchronization at play-out time. It thus becomes much easier for authors 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 *trigger condition*, expressed in terms of state variables in the source objects, triggers execution of a link object. 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, the condition for executing action objects will be checked at runtime 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 termination of both an audio sequence and a parallel video sequence. In this case the trigger condition is a conjunction of two conditions bound to different source objects. You can add such conditions to the firing of a link using the constraint condition statement in MHEG. Such a statement expresses a required contextual state at the moment when the trigger condition is satisfied. If

there were no two-source conditions, the author would have to define both sequences of events separately, that is, audio terminates first or video terminates first. This would lead to the somewhat unnatural definition of two different trigger conditions in the same link object.

The trigger and constraint conditions both evaluate Boolean values. You can 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 (see Figure 8), we used 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 must be encoded for each interaction (denoted by arrows in Figure 4) between objects in a presentation.

MHEG not only makes it possible to trigger based on certain status fields, it also permits triggering based on parameter changes of rt-objects. For example, you can define a link dependent on the adjustment of the volume parameter attached to an audio object. Furthermore, a macro facility provides reuse of complex action objects. A macro is comparable to the well-known `#define` clause in C preprocessors. Parameters of elementary actions can be changed at runtime before the action goes to the target object. This generalization of the primitive link model helps authors compose more effective presentations.

User interaction

With the content class, the action class, and the link class already introduced, 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. "Selection" allows simple user interaction from a predefined set of alternatives, whereas "modification" permits more complex data input.

Selection. The discussion of the link class showed how the author can influence the play-out of a presentation based on events occurring at a predefined point in time. With selection behavior, the user can create such events at runtime. To define such events, the author assigns for each possible interaction a corresponding internal event. When the user interacts with the system—by pressing a button, for example—the corresponding selection status changes from Not-Selected to Selected and the assigned value is set to the attribute "selection mode." Changing the selection status can trigger a link object and prompt the selection mode to evaluate the constraint condition. The selection behavior is associated with an rt-content object, for example a graphic on the screen. The author can also define hierarchical selections such as pull-down menus.

```
Link-Class {
  MHEG-Identifier.Object-Number: 4,
  Link-Condition {
    Source-Value.Get-Modification-Status.Object-Number: 10,
    Previous-Condition.Evaluated-Condition {
      Comparison-Operator: equal,
      Comparison-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
}
```

Figure 8. Each case of object interaction requires a link object, as shown in this ASN.1 example.

Modification. The second and more general form of interaction in MHEG is modification behavior, designed for the entry and manipulation of data. In contrast to selection behavior, 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 stores 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. You could thus, similar to selection, include a modification status in the condition of a link object and 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 behaviors of an rt-content object. Interaction behavior defines the style of these kinds of user interaction. MHEG contains five predefined styles: button, slider, entry field, menu, and scrolling list. For example, the action "set-button-style" will attach the selection status to a graphic object to behave as a button, and "set-entryfield-style" attaches the modification status to text to perform

as an entry field. An author can define the look and feel of the user interaction or simply use primitives provided by the graphical user interface of the executing MHEG engine.

The following actions encode the selection of 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 when 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. It represents the number of the video the user wants to see.

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

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 the interchange of these objects separately, a data structure is needed to compose the multimedia presentation.

The composite class. The composite class gives the skeleton of a presentation. 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 composite objects to be part of other composite objects.

From a logical point of view a composite object can be understood as a kind of container. Comparable to the content class, in the composite class MHEG distinguishes between included or referenced objects. External references help split up a complex presentation into smaller pieces to reduce the amount of memory needed in the end system. They also support real-time requirements by providing partial object retrieval. For example, references between composite objects set up a hypertext-like structure. The execution environment is responsible for establishing access to referenced objects.

The following rudimentary encoding refers to the initial composite object of the scenario. The Predefined-Behavior specifies default link objects with implicitly defined trigger conditions. 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 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 action 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 on how to reconstruct the presentation.

The descriptor class encodes information about a presentation's objects, 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. An MHEG engine can use this information to evaluate the presentation's requirements with respect to the runtime platform's available resources.

The script class provides an interface to external functions or programs. A typical application is a database query. The query input is encoded by suitable MHEG objects, whereas a script object invokes the query executed in the script environment and transmits data between the MHEG engine and the script environment. For all the details of these two classes, refer to the MHEG document.[8]

MHEG implementation issues

The standard describes the MHEG object structure and transfer encoding in great detail, but says little about the implementation of an MHEG engine. Early experience with MHEG implementation environments comes from 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. After looking at the architecture of the MHEG runtime environment, we move to playback at the presentation terminal.

Runtime environment architecture

Figure 9 presents the prototype developed for a networked multimedia kiosk environment. This environment consists of presentation terminals running the MHEG engine. These terminals connect to a remote database server to retrieve MHEG-encoded interactive multimedia presentations. An authoring workstation provides an interactive editor for composing MHEG objects. The implementation of the prototype, based on the committee draft, is now available under IBM AIX for RS/6000. An OS/2 version based on the Draft International Standard is under development.

Recently researchers in France[10] and in Korea[11] presented similar MHEG runtime environments. Both prototypes were implemented as distributed retrieval systems for Microsoft Windows 3.1. The Korean implementors reported problems with the nonpreemptive 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 implementing an MHEG engine in an operating system that supports preemptive multitasking.

Authoring a 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. Each object must be in one of the standardized formats supported by the MHEG runtime environment. The author composes the MHEG presentation from these content objects.

Authoring tools (such as Macromind Director) are already available on the market. Most have their own script language (IBM's AVA/2, Kaleida's Script/X) supporting the author with application-oriented semantics. Authors could also 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 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[12]). Converting script languages to MHEG can pose some hard problems because MHEG's link model cannot express loops.

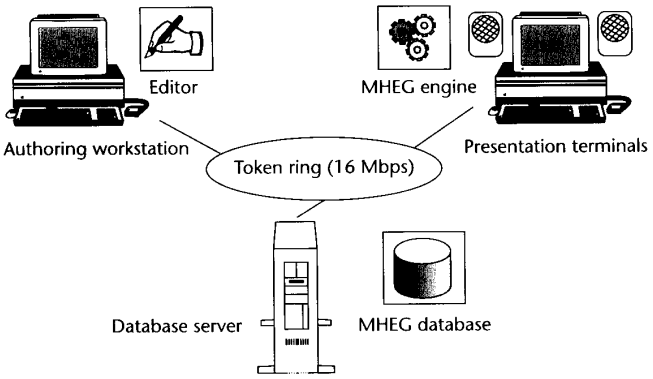

Figure 9. This diagram shows the setup for our MHEG-based networked kiosk system.

As mentioned, the standard representation of MHEG objects is in ASN.1. This language has a very low level of detail, and we do not recommend authoring sophisticated multimedia presentations directly at this level. Doing so 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 low level.

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

Distribution of MHEG objects. Once a presentation is completely encoded, it can be transmitted through all forms of information exchange. Typical examples are 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 works. A comparable approach is under development by Kaleida. The Script/X authoring environment also supports a lower level interface, the Script/X Byte Code.[13]

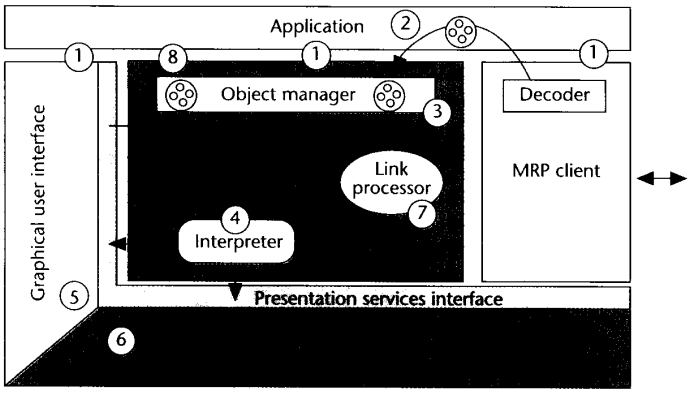

Figure 10. MHEG objects flow through the components of an MHEG engine in eight steps, listed in the text.

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.

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, handle MHEG events and links, and execute the interpreters for all MHEG's basic content object types. The author has programmed possible user interactions using the selection and modification classes, and interactions are only possible at points preplanned by the author. Figure 10 shows the flow of MHEG objects through the components of an MHEG engine.

Let us now consider the steps taken during playback of the MHEG-encoded kiosk presentation, first briefly, then in more detail:

1. The application initializes the MHEG engine, MRP client, and presentation services. The first object is retrieved and prepared.
2. The application starts the presentation by sending the root object to the MHEG engine.
3. The object manager stores MHEG objects loaded into the engine during execution.
4. The interpreter is the heart of the MHEG engine and processes all events.
5. The presentation services implement the rt-content objects for all discrete and continuous media types.
6. An audio/video subsystem provides a real-time environment to guarantee timely and synchronized playback of continuous media.
7. If the conditions of a link object are fulfilled, then the link processor sends the actions defined in the link object to the target object via the central message queue.
8. After all composite objects have been executed, the MHEG engine frees local resources and stops.

Step 1. The application initializes the MHEG engine, the MHEG request/response protocol (MRP) client, and the presentation services. The latter typically consist of the local audio and video drivers and the window-based user interface. Initialization prepares the first object of the MHEG presentation for execution, 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.

Step 2. The MRP client provides network-transparent access to all MHEG objects. Before they can be interpreted, the objects must be decoded from the transfer syntax (Basic Encoding Rules for ASN.1) into the local syntax. After the transfer completes, the application starts the presentation by sending the root object to the MHEG engine.

Step 3. The object manager is a central resource that stores all MHEG objects loaded into the engine during the execution cycle. Before they can be interpreted, the objects must be decoded from the transfer syntax into the local syntax. If an MHEG object contains a reference to another object, the object manager also requests the transfer of that object.

Step 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.

Step 5. During interpretation the MHEG interpreter calls the presentation services, which implement the rt-content objects. For example, the

action Run on a text object will show that object's data on the user's screen. Discrete media (text, graphic, etc.) and primitives such as sliders, buttons, and entry fields provided by the graphical user interface are executed in the non-real-time environment of the presentation services.

Step 6. An audio/video subsystem with corresponding drivers provides a real-time environment where scheduling policies guarantee synchronized playback of continuous media. Typically these drivers directly support the basic content types defined in the MHEG standard, such as MPEG video or PCM-encoded audio. Often these media streams are not contained in the MHEG objects but are referenced. In this case the digitized audio or video data have to be retrieved from a multimedia file system or from a remote site via a multimedia transport system.

Step 7. When an incoming message is processed, the various status fields of the interpreter can change. Examples include 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 link processor sends the actions defined in the link object to the target object by adding them to the central message queue.

Step 8. After all composite objects of a presentation have been executed, the MHEG engine frees all local resources and is ready to start a new presentation.

Note that the pending 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. These are considered implementation issues. For specific multimedia applications, such interfaces are currently being defined (see, for example, ITU Recommendation T.170[14]).

Conclusions and outlook

Until today, research and development have 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 standardized formats not only for media contents, but also for the structure of a complex, interactive multimedia presentation. The forthcoming MHEG standard will provide these formats. It will allow distribution of complex multimedia information in final form and playout on heterogeneous systems. This technology will enable a large number of new applications.

The standardization process is now reaching the draft international standard 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 come the following contributions: 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 objects, logical combination of conditions represented as a tree structure, and support of a group concept to synchronize multiple continuous streams stored in individual content objects.

We hope that the exchange of complex multimedia information between heterogeneous systems will soon be a reality. The German project Globally Accessible Services, sponsored by DeTeBerkom, is developing a DIS-based version of the presentation terminal for various platforms. Figure 11 shows an example of a screen produced with MHEG on that system. We expect to see other such implementations soon. **MM**

Figure 11. An example of a screen produced with MHEG on the Globally Accessible Services' DIS-based presentation terminal.

Acknowledgments

The implementation project described in this article took place at IBM's European Networking Center in Heidelberg. We would like to thank IBM for the financial support and our colleagues at ENC

for their excellent cooperation. Special thanks go to Ralf Steinmetz for his helpful comments on an earlier version of the article and to Guido Grassel for coding many parts of the prototype.

References

1. ISO/IEC IS 10918:1992, "Information Technology Digital Compression and Coding of Continuous-Tone Still Images (JPEG)."
2. 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)."
3. F. Kretz and F. Colaitis, "Standardizing Hypermedia Information Objects," *IEEE Comm. Magazine*, Vol. 1, No. 5, May 1992, pp. 60-70.
4. R. Price, "MHEG: An Introduction to the Future International Standard for Hypermedia Interchange," *Proc. 1st ACM Int'l Conf. on Multimedia*, ACM Press, New York, 1993, pp. 121-128.
5. ISO/IEC IS 8824:1987, "Information Processing Systems Open Systems Interconnection Specification of Abstract Syntax Notation One (ASN.1)."
6. 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."
7. W. Holfelder and D. Hehmann, "A Networked Multimedia Retrieval Management System for Distributed Multimedia Kiosk System Support," *Proc. 1st IEEE Conf. on Multimedia Computing and Systems*, 1994, CS Press, Los Alamitos, Calif., pp. 132-143.
8. ISO/IEC CD 13552-1:1993, "Information Technology Coded Representation of Multimedia and Hypermedia Information Objects (MHEG) Part 1: Base Notation (ASN.1)," Oct. 15, 1994.
9. N.S. Borenstein, "MIME: A Portable and Robust Multimedia Format for Internet Mail," *ACM Multimedia Systems*, Vol. 1, 1993, Springer, New York, pp. 29-36.
10. C. Bertin, "Eurescom IMS1 Projects (Integrated Multimedia Services at about 1 Mbit/s)," *Proc. 2nd Int'l Workshop on Multimedia: Advanced Teleservices and High-Speed Communication Architectures*, R. Steinmetz, ed., Springer Lecture Notes in Computer Science, Vol. 868, Berlin, 1994, pp. 53-66.
11. J.-J. Sung et al., "Hypermedia Information Retrieval System Using MHEG Coded Representation in a Networked Environment," *Proc. 2nd Int'l Workshop on Multimedia: Advanced Teleservices and High-Speed Comm. Architectures*, R. Steinmetz, ed., Springer Lecture Notes in Computer Science, Vol. 868, Berlin, 1994, pp. 67-77.
12. B. Markey, "Emerging Hypermedia Standards: Hypermedia Marketplace Prepares for HyTime and MHEG," (U.S.) Assistant Secretary of Defense (Production and Logistics), Washington, D.C., PB92-120328, 1991.
13. Kaleida Labs, "Kaleida, Script/X, and the Multimedia Market," Kaleida White Paper, Revision 1, Mountain View, Calif., 1994.
14. ITU-T Draft Recommondation T.170, "Audiovisual Interactive (AVI) Systems General Introduction, Principles, Concepts, and Models," Second Revision, Geneva, Nov. 16-25, 1993.

Thomas Meyer-Boudnik is a technical consultant with Vebacom GmbH, a German telecommunications provider, in the Corporate Networks/Value-Added Services department. His interests are distributed multimedia systems and value-added service for broadband networks. He received his diploma in computer science and business administration from the University of Mannheim, Germany, in 1991, where he is currently finishing his PhD in computer science.

Wolfgang Effelsberg is a professor of computer science at the University of Mannheim, where he teaches about computer networks, distributed systems, and multimedia technology. He received his diploma in electrical engineering and Dr.-Ing. degree in computer science from the Technical University of Darmstadt, Germany, in 1976 and 1981, respectively. He is a member of ACM, IEEE Computer Society, and Gesellschaft für Informatik.

Readers may reach Meyer-Boudnik at Vebacom GmbH, Am Bonneshof 35, 40474 Düsseldorf, Germany, and Effelsberg at the University of Mannheim, L 15, 16, 68131 Mannheim, Germany. Effelsberg is reachable by e-mail at effelsberg@pi4.informatik.uni-mannheim.de.