ELECTRONIC PUBLISHING, VOL. 7(3), 179–193 (DECEMBER 1994)

Using the MHEG standard in the hypermedia system Multicard

ANTOINE RIZK AND FRANCIS MALEZIEUX

Euroclid, Promopole
12 Avenue des Prés
78180 Montigny-le-Bretonneux
France

ALAIN LEGER

CCETT
4 Rue du Clos Courtel
35512 Cesson Sévigné
France

SUMMARY

The MHEG standard will define a coded representation of multimedia and hypermedia information objects so as to facilitate exchange of hypermedia applications over various platforms. This standard has been developed entirely independently of existing architectures such as Dexter and 'Dexter like' systems such as Multicard, KMS[1] etc.

In order for the MHEG standard to succeed, it is important that existing hypermedia systems and applications can be rendered MHEG compatible, rather than these applications having to be rewritten using new MHEG engines.

This paper provides a case study of how the MHEG standard could be adopted in one such hypermedia system, namely Multicard. The aim is to highlight the similarities and differences of the MHEG standard and Multicard and to provide an idea of the work required in order for such a system to read MHEG compatible streams. The paper starts with a brief description of the Multicard system, the Dexter model and the MHEG standard.

KEY WORDS MHEG Multicard Hypermedia

1 INTRODUCTION

The international standard being developed by the Multimedia Hypermedia Experts Group of ISO, known as the MHEG standard [2–5], will define the representation and encoding of multimedia and hypermedia objects to be interchanged within or across applications or services, by any means of interchange including storage devices, telecommunications or broadcast networks.

These objects, encoded using ASN.1 or SGML, will provide a common base for other CCITT recommendations and ISO standards, and for the many multimedia and hypermedia applications which will be developed in the future in a wide range of domains. The MHEG specification addresses the needs of minimal resource terminals and makes use of other standards for the component text, image, graphic and other objects.

Multicard [6] is a complete hypermedia platform whose architecture is akin to that of Dexter [7]. It comprises a set of hypermedia basic classes (nodes, links, groups...) with their associated API and persistent object storage, a scripting language, a multimedia composition editor and a communication protocol (M2000) that allows any compliant editor to be used with the contents of Multicard nodes.

The aim of this paper is:

- to determine the internal representation of the MHEG classes in terms of Multicard hypermedia basic classes;

CCC 0894–3982/94/030179–15

Received 15 May 1993
Revised 15 October 1993

- to determine how the MHEG attributes map into Multicard;
- to define the conversion structures from MHEG objects to Multicard ones and vice versa.

The MHEG standard defines object classes that correspond to multimedia/hypermedia information units, with the unique goal of favouring their exchange. It does not specify how MHEG engines, interpreters or any kind of MHEG application should be designed around these classes. Neither does it specify the internal representation of the classes. It is, therefore, only with respect to exchanging instance objects of MHEG classes that we will speak hereafter of conformance to MHEG.

The paper is structured as follows: Section 2 provides a brief overview of the MHEG standard. Section 3 describes the Dexter model and Section 4 the Multicard architecture. Section 5 is the actual comparison of Multicard and MHEG, and Section 6 is an example transformation of a Multicard hypergraph into a MHEG structure.

2 THE MHEG STANDARD

The initial objectives of the MHEG standard—'Defining and providing abstractions for multimedia and hypermedia applications'—have been set forth in the following requirements:

- Provide abstractions suited to real-time presentation: this real-time requirement is fulfilled by multimedia synchronization functionalities.
- Provide abstractions suited to real-time interchange: this means interchange with minimal buffering using normal speed data communication.
- Provide abstractions corresponding to a final form representation: the objects are represented and coded with the aim of a direct presentation, without requiring an additional processing of their structure.

2.1 MHEG: a standard which defines objects

The object-oriented approach was chosen for the design of the standard because it fits the requirements of active, autonomous and reusable objects. The standard defines classes of objects, the design of which relies on the analysis of their common behaviour and the commonalities of properties between object categories. It provides a description of each class, a precise definition of the representation of the MH objects, which are instances of the classes, and a coded representation for the objects (base: ASN.1, alternative: SGML).

The MHEG standard makes a distinction between an interchanged object—which contains the structural information, this is the original reusable object—and a 'viewer' of this object, which corresponds to a specific 'view' of the object at the presentation time: the viewer presentation does not affect the original object.

The object classes and their actual instances are represented through an inheritance tree (see Figure 1).

Directly derived from the inheritance tree, the 'tool-box' for the design of multimedia and hypermedia applications is the following:

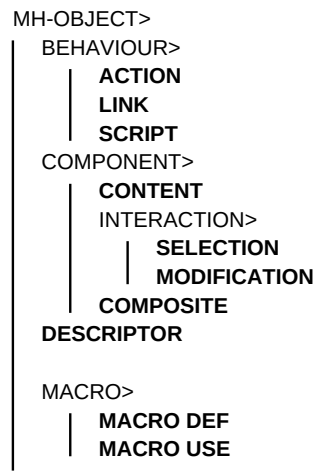

Figure 1. MHEG inheritance tree

1. *Content class*
   It provides for the encapsulation of coded content data (i.e. an image, a piece of sound, a graphic, etc.) associated with public or private decoding specifications.
   It is the basic element on which spatio-temporal relations (link class), hypertext-like relations (link class), and a set of actions are applied (action class), e.g. Run ObjectId 1022, Destroy ObjectID 14, etc.
2. *Link class*
   It provides a generic linking mechanism for both multimedia and hypermedia applications.
   The link class provides the following information for the interchange of conditional actions to be applied to instances of any MHEG class and/or viewers:
   - a set of one-way links;
   - a list of triggering conditions;
   - a list of actions to be applied to the destination, if the previous conditions are satisfied.

   This is illustrated by Figure 2.
3. *Action class*
   The standard provides three types of actions than can be applied on MHEG objects:
   - Actions that may affect the status of objects: this sort of action, such as 'prepare', 'run', 'destroy', will affect the state of the object to which they are applied, for example an object with the state 'not ready' will become ready after the completion of a successful 'prepare' action. The state transitions can be used as a triggering condition to fire the link.
   - Actions for the projection of objects: to define precisely the way the objects have to be presented. For example, 'set volume', 'set visible size', 'set position x,y', are some of them.

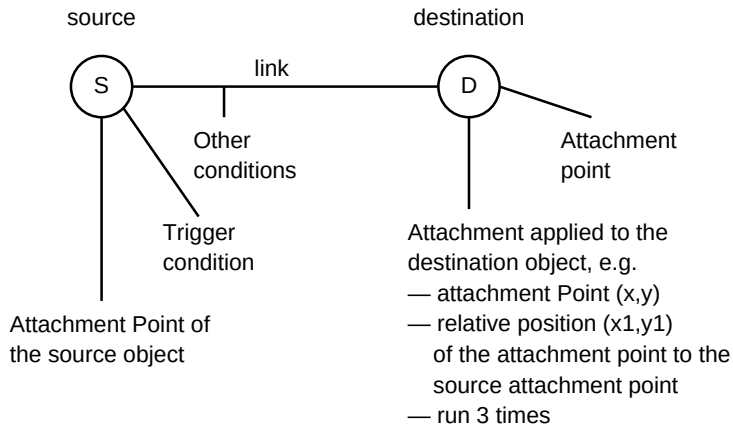

Figure 2. Illustration of link and actions

- Actions that may affect the interaction on objects: these actions are used to modify specific interaction parameters of the interaction class. For example the number of menu items, or those that are selectable at a given time, or if the content data of a content object is modifiable or not, etc.

4. *The interaction class*
   This class provides for the support of the interaction with the user. It provides all the tools at the MHEG level to make the document 'reactive' to the user.

   Nevertheless, MHEG does not define the 'look and feel' of the multimedia interactive presentations, neither does it propose to change or add concepts to those that exist in typical graphical user interfaces. As this standard is generic and independent of platform and implementation, it describes interaction at a logical level. It is for the using application to apply these mechanisms using its specific 'look and feel'.
5. *The composite class*
   This class mainly aggregates the preceding tools. It is a container used to interchange a set of inter-related objects (spatio-temporal links, 'hypermedia' links, and actions).

3 THE DEXTER MODEL

The Dexter model [8] is a reference model that was designed to serve as a basis for interoperability and information exchange amongst applications. Although these objectives were not achieved, the general philosophy of Dexter was inspired by, and now reflects, most classical hypertext systems.

The model (see Figure3) consists of three layers and two interfaces:

- The run-time layer describes the interaction mechanisms of the hypermedia system with the user.
- The storage layer describes the nodes, and the links, as well as the associated relations for constructing a hypergraph. In this layer, nodes are treated separately from their contents.
- The within-component layer describes the node structure and contents.

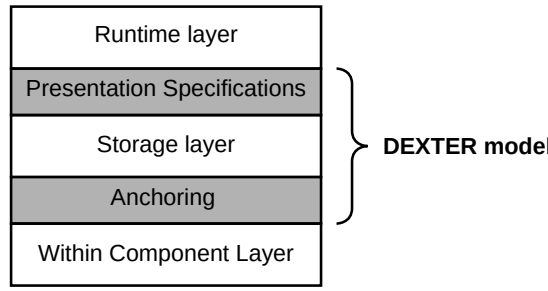

Figure 3. Layers of the DEXTER model

The DEXTER model focuses on the storage layer but underlines the importance of the interfaces with the other two layers.

The first of these interfaces, anchoring, provides the mechanisms for anchoring the links in a portion of the contents. It is therefore essential for addressing link end-points.

The second interface, presentation specifications, provides the principal mechanisms for specifying how nodes, links and contents could be presented for interaction with the user.

The importance of DEXTER is that it provides a decomposition of the essential components of a hypertext system. Existing systems do not implement the exact specifications provided by DEXTER but most of them relate to the model or to subcomponents of it.

4 THE MULTICARD ARCHITECTURE

The Multicard hypermedia system architecture, illustrated in Figure 4, is represented as a set of components and interfaces. The architecture is based on the general model of front-end and back-end subsystems. It has the following distinct layers: a set of hypermedia basic classes that constitute the toolkit; hypermedia distributed persistent object storage; an authoring/navigation tool; a communication protocol; and a series of compliant editors.

4.1 Hypermedia objects

The heart of the hypermedia toolkit is the representation of hypermedia objects (nodes, groups, anchors and links, hypergraph, etc.) together with the associated interfaces to applications, the scripting language and the editable objects. These hypermedia objects are implemented in C++. They can be accessed either from C++ or through a C binding. We recount here, briefly, the specific features of these objects:

Nodes: In Multicard, there is a difference between node structure, which manages links scripts, and the content of the node. Multicard manages the node structure, whereas the contents of the node may be handled by different editors. The M2000 protocol between Multicard and the editors allows the opening and closing of documents, retrieval of content portions, etc.

Anchors: An anchor represents a sensitive portion of the content of a node. The associated anchor is the hypermedia object that carries the links, scripts, and other hypermedia properties. The sensitive portion is editor-dependent.

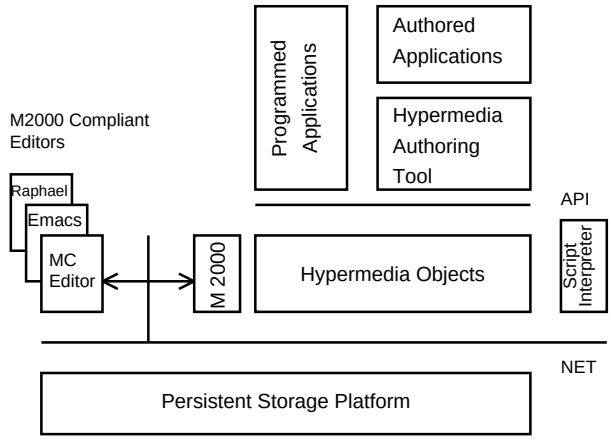

Figure 4. Multicard architecture

Groups: Groups represent logical collections of nodes and other groups. Group hierarchy can be of unlimited depth.

Links: Contrary to the usual usage of links in hypertext systems, links in Multicard are viewed as event/message communication channels between two end points. Various messages can be sent through a link including, of course, the activation message which will typically open and map the destination object. Link end-points can be anchors, nodes or groups.

The Application Programming Interface (API) provides facilities to specific hypermedia editors, tools, and general applications for the creation, manipulation and deletion of hypermedia objects. The M2000 interface supports interaction between hypermedia objects and the relevant content-based editors.

4.2 Script interpreter

Nodes, groups and anchors may have scripts attached to them. In this sense scripts are used as a way of extending the behaviour of instances of these objects. Scripts are event-driven and can communicate throughout the hypermedia application using event/message passing. The scripting language contains over 150 instructions that bring the API closer to the end-user, and include special instructions for manipulating editor contents, synchronizing with the editor, definition of internal/external functions, etc.

4.3 Hypermedia persistent storage

This provides distributed persistent storage for hypermedia basic objects and consists of two parts. The front-end supports access by the hypermedia basic class objects and guarantees consistent behaviour independently of the actual storage management implemented by the back-end. This approach enables the storage mechanism to be implemented using relational or object-oriented databases [9,10] without affecting the toolkit interface.

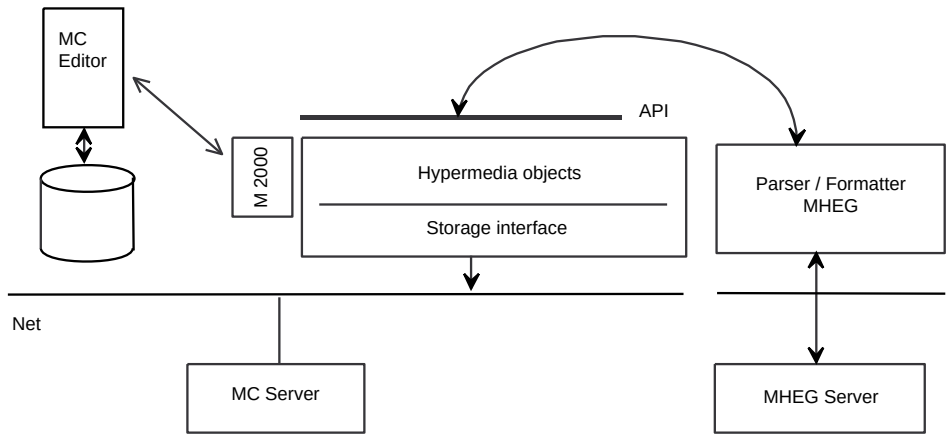

Figure 5. MHEG application using the Multicard API

5 MULTICARD AND THE BASIC MHEG CONCEPTS

In this section we look at the principal concepts of the MHEG standard and try to position Multicard with respect to these concepts. We will show that, as far as essential concepts are concerned, for the exchange of multimedia and hypermedia information, these concepts are present in Multicard. As we have already mentioned, it is only in view of exchanging instance objects of MHEG classes that we speak here of conformance to MHEG. One possible scenario for the integration of MHEG in Multicard is shown in Figure 5, whereby an MHEG stream is parsed in input (or output by the formatter), and translated into Multicard hypermedia basic classes using the toolkit API.

5.1 Synchronization

MHEG recognizes four kinds of synchronization:

- At the script level, MHEG underlines the importance of scripting standards such as the AVIs [11,12]. However, MHEG does not define such synchronization but rather provides pointers in its classes to scripting standards.
- Conditional synchronization, in order to relate actions to events produced by other actions, by the system or by the user.
- Synchronization in space/time, that allows objects to be composed within a space with coordinated temporal behaviour.
- System synchronization, which is often intrinsic to the manipulated objects, as for example is the case for MPEG objects that possess their own audio/video synchronization. This form of synchronization is not addressed in MHEG.

In Multicard, as in many other hypermedia systems, the scripting language is fundamental to the definition of instance behaviour. The scripting language is event-driven. As such, scripts are event handlers that could only be activated on occurrence of certain events, triggered by the system, the user, or by any other script activation.

This kind of scripting covers the first two types of synchronization above, namely the script level and the conditional synchronization. Space/time synchronization is also

possible, but at a 'loose', non-real-time, level. Real-time space/time synchronization as well as system synchronization are considered in Multicard as pertaining to the editor that is responsible for content manipulation.

As a consequence, synchronization relations, expressed by MHEG in links, composite objects, interaction and action objects, are translated systematically into Multicard scripts.

5.2 Links

Two types of link exist in MHEG :

- The external reference links, provided as ISO 9070 identifiers.
- The specific inter-object links, provided for synchronization as well as for direct access to objects.

The first type of link is rather an 'in stream' identifier to an external, remote object. The interpretation of this is application-dependent, and might or might not mean a hypertext goto link.

In Multicard, links are objects in themselves that correspond more closely to the second type of link. Links will carry, through their source and destination anchors, scripts that correspond to the actions defined in the behaviour of the MHEG link.

An instance of the MHEG link class consists, on the one hand, of an activation condition and supplementary link invocation conditions and, on the other hand, of a series of actions to be carried out on one or more referenced objects. This series of actions is defined in an object instance of the Action class.

A Multicard link is a message channel that has as a default action the activation of the link destination. It is therefore necessary to use a link+script combination in order to translate an instance of an MHEG link. For each such link instance, we proceed in Multicard in three steps:

- create a link from the source object to every destination object;
- create, on the source of the link, the trigger handler responsible for the activation of the links;
- create the action handler for each destination object.

5.3 Content/structure separation

MHEG, as in Multicard and Dexter, defines a strict separation of the container objects that MHEG recognizes and the application that recognizes the content format. Being mainly a non-processable format, MHEG does not define the interaction interface between the content editor and the object management. Moreover, anchors in MHEG are not defined as subsets of the content format but rather as presentation objects (buttons etc.) that coincide with specific contents.

5.4 Composition

Composites are introduced in MHEG in order to simplify the management and presentation of sets of objects. Composite objects in MHEG contain the component objects themselves

or references to them, as well as the relations that unite these objects. Composite objects are recursive.

Composite objects in MHEG always resolve finally to basic content objects.

In Multicard, as in Dexter, composites are sets of nodes/composites at the abstract level. This composition is used for managing large numbers of nodes rather than for multimedia composition at presentation time. Multimedia composition is done exclusively by the editor that manages the contents of the node. The content objects that the editor manipulates and composes have no representation in the hypermedia basic classes.

Composition in MHEG covers these two fundamentally separate concepts in one, by specifying the semantics of the composite in the associated actions. In order to translate an MHEG composite object to Multicard, an interpretation of the actions is required. Depending on this interpretation, the composite is translated to a Multicard group (Dexter composite of nodes) or to a node with MCEditor (Multicard's multimedia composition editor).

5.4.1 The content class

An instance of the content class is assimilated to a node instance in Multicard. An instance of the content class comprises principally:

- An attribute that describes the coding of the object (hook). This is interpreted as the creation of a corresponding editable object, and when the corresponding editor does not exist, format conversion is attempted.
- Either data encapsulation or an attribute identification of contents. In the case of data encapsulation, an editable object is created, whereas, in the case of external identification, Multicard has to rely on external services for the localization and migration of the remote object.

5.4.2 The selection class

This class defines the usage of selection elements such as menus, buttons and sliders, as well as all GUI predefined widgets, for user interaction. A particular instance of this class could also specify a particular selection.

In the first case, Multicard does not offer an equivalent mechanism, it simply relies on the X-Window/Motif predefined widgets. In the second case, particular selection positions correspond to anchors in Multicard that are attached to sensitive zones in the contents of editable objects.

The response of selected elements is defined using the Multicard scripting language.

5.4.3 The modification class

An instance of this class simply specifies whether the content object it refers to is modifiable or not. In Multicard this is simply implemented as an additional attribute to the editable object classes as well as hypermedia basic classes. It can be set/reset dynamically by script.

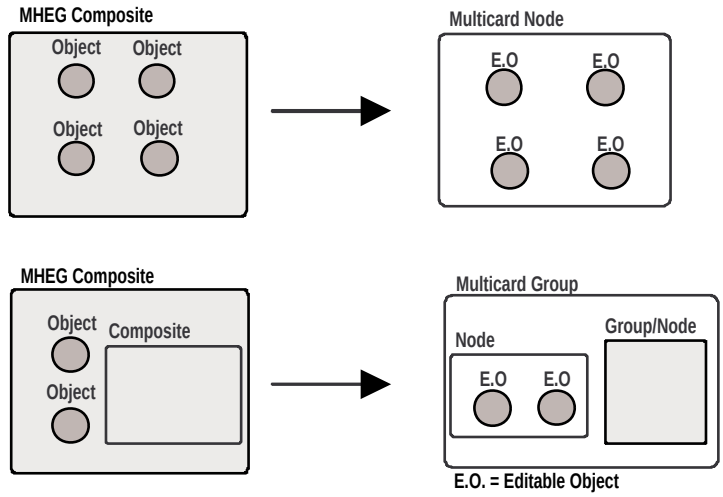

Figure 6. Composition in Multicard

5.4.4 The composite class

An instance of the composite class defines a set of links, the associated actions, and a set of objects (or references to objects) on which these actions apply. A start up link specifies the actions to be executed upon activation of the composite object.

In addition a viewer link is specified. This is used at activation of the composite object and defines the projection of the object in space. In Multicard, this viewer is defined as a set of parameters that are subsequently interpreted by a generic handler that sets the editable object properties accordingly.

The representation of MHEG composition in Multicard draws upon the group and node classes. It is necessary to use both of these hypermedia classes as the MHEG composite is at the same time a composition of abstract node objects in the Dexter sense and a physical composition of content objects in the multimedia sense.

Multimedia composition in Multicard can only be done using the MCEditor. This editor can only compose editable objects within the contents of a single node. However, the node is not a composition object in its own right since it cannot contain other nodes. Moreover, the composition object in Multicard can only compose other nodes or groups but not editable objects.

Therefore, composition in Multicard is done by using both nodes and groups. During translation, an MHEG composite is translated into a Multicard group if it contains other composites, and to a Multicard node if it contains only content objects. Figure 6 illustrates how MHEG composition is done in Multicard.

5.5 MHEG and Multicard classes

Figure 7 depicts the basic class hierarchy, both for MHEG and Multicard.

We note the following:

1. The notion of interaction objects does not exist in Multicard. An interaction object in Multicard is an ActiveObject that possesses its script. The mapping between the ActiveObject and a real presentation is done by attaching an anchor object on an editable object such as selector and modifier.

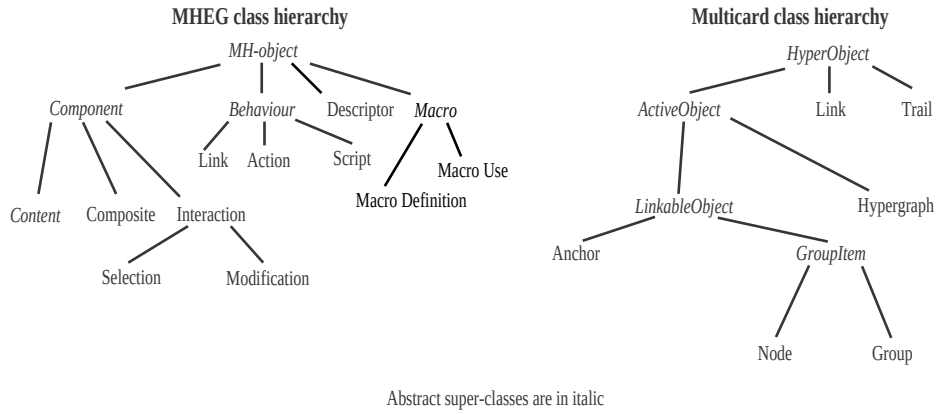

Figure 7. MHEG and Multicard class hierarchies

2. The script class does not exist as such in Multicard. Rather, a script is an attribute to the class ActiveObject.
3. The descriptor class does not exist in Multicard. It has to be interpreted using scripts. The notion of presentation attributes that exist in MHEG is, however, available clearly in Dexter through the run-time interface layer.
4. Composition could not be mapped on a one-to-one basis. Depending on the associated actions, composition is mapped using GroupItem, Group and Node.

The following table illustrates, one by one, the mapping of MHEG concepts onto Multicard ones.

MHEG	Multicard
MH_Object Class	HyperObject
Descriptor Class	To be interpreted
Script Class	Attribute Script of ActiveObject
Content Class	Node
Composite Class	Node, Group and GroupItem classes
Link and Action Classes	Link and scripts associated to objects (source and destination)
Interaction Class	Anchor, editable objects and GUI

6 EXAMPLE: CONVERSION OF A MULTICARD STRUCTURE

Consider a small Multicard document whose hypergraph is represented as follows.

There are two possible ways for translating this structure to MHEG.

The first solution is illustrated in Figure 9. This solution uses an MHEG link object to map the Multicard group (which is a purely logical composition). Each node in Multicard is translated to an MHEG composite object. Navigational links that are used in Multicard are mapped into MHEG links.

The second way of proceeding uses an MHEG composite object to map the Multicard group (Figure 10). Depending on their complexity, nodes appear here in a single MHEG composite.

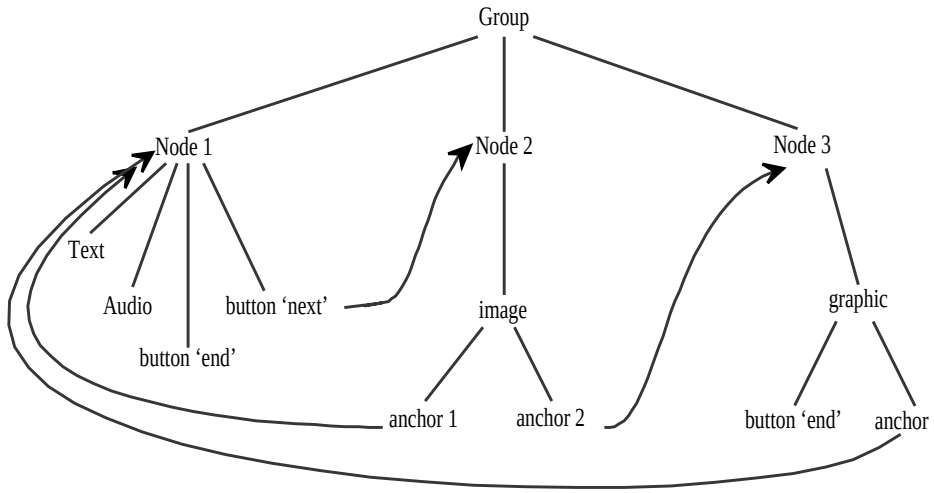

Figure 8. Hypergraph of a small Multicard document

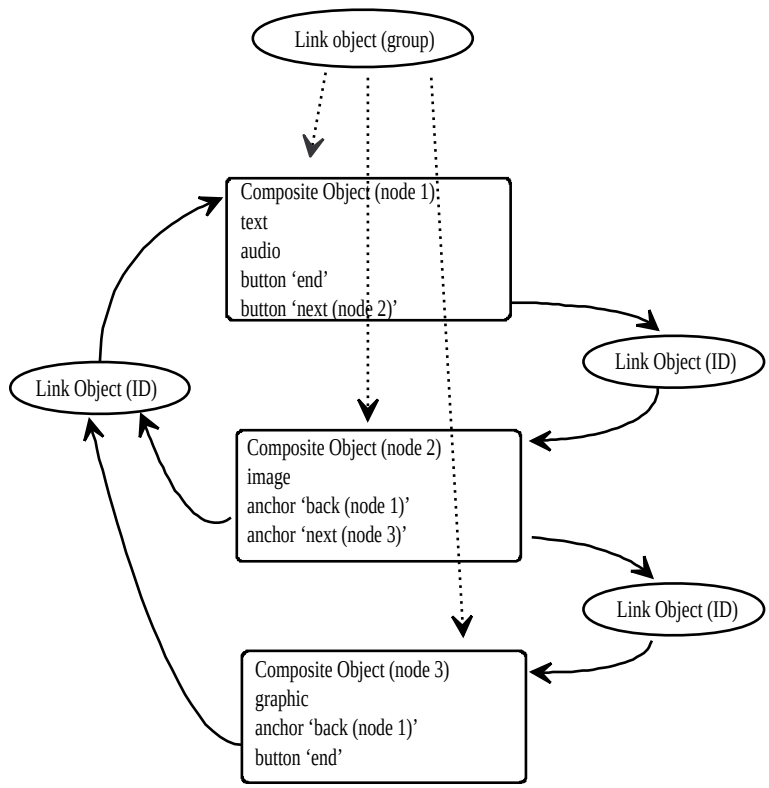

Figure 9. Description with MHEG composites

Composite(group)
All-object representation
attributes
interchange number = xy

- Composite start-up link:
  destination: viewer "node 1", action: prepare
  destination: viewer "node 2", action: prepare
  destination: viewer "node 3", action: prepare

- Viewer start-up link:
  destination: viewer 1, action: present "node 1"
  destination: viewer 2, action: present "node 2"
  destination: viewer 3, action: present "node 3"
  destination: viewer 4, action: present "anchor 1"
  destination: viewer 5, action: present "anchor 2"
  destination: viewer 6, action: present "anchor 3"
  destination: viewer 7, action: present "button next"
  destination: viewer 8, action: present "button end"

- Links:

S: node 1, "button next",	D: viewer "node 2"
Trigger: "button next" selected,	Action: go to
S: node 2, component "image", anchor 1,	D: viewer "node 1"
Trigger: anchor 1 selected,	Action: go to
S: node 2, component "image", anchor 2,	D: viewer "node 3"
Trigger: anchor 2 selected,	Action: go to
S: node 3, component "graphic", anchor 3,	D: viewer "node 3"
Trigger: anchor 3 selected,	Action: go to

- Viewers:

viewer "node 1"	= component 1
viewer "node 2"	= component 2
viewer "node 3"	= component 3
viewer "anchor 1"	= component 4
viewer "anchor 2"	= component 5
viewer "anchor 3"	= component 6
viewer "button next"	= component 7
viewer "button end"	= component 8

- Components:
  component 1 = ref. to composite "node 1"
  component 2 = ref. to composite "node 2"
  component 3 = ref. to composite "node 3"
  component 4
  description of "anchor 1" (included)
  component 5

etc. . . .

Figure 10. Description with one global MHEG composite

7 CONCLUSIONS

This paper provided a case study of how the MHEG standard could be adopted in an existing hypermedia system, namely Multicard. To our knowledge this work is so far the first attempt of its kind. Although the implementation of the full MHEG–Multicard converter is not yet complete, the concepts presented have been subject to specific validation tests.

From our evaluation of the MHEG standard, Multicard, and more generally Dexter, we observe that although MHEG has been thought out entirely independently of Dexter, as far as the fundamental concepts are concerned, the two models are extremely similar. We find the presence of separation of contents from hypermedia structure, composition, links and presentation specifications.

On the more detailed level, MHEG differs from Dexter on the following points. The anchoring layer of Dexter is extremely weak in MHEG, due basically to MHEG being an unprocessable form. The composition in MHEG can deal both with multimedia composition including space/time synchronization of objects, as well as the Dexter abstract composition of nodes. As far as implementation of MHEG by an existing hypermedia system, it is essential that the system possesses a powerful scripting language for the interpretation of the many dynamic attributes, as well as for the interpretation of the MHEG actions.

ACKNOWLEDGEMENTS

The authors wish to thank the ISO-JTC1 SC29/WG12 (MHEG) members and CCITT SGVIII (terminals and protocols) for their invaluable work that made the present effort possible. The constructive and sometimes provocative comments of Françoise Colaïtis (MHEG convenor), Florence Bertrand and Roger Price (MHEG standard editors) are also gratefully acknowledged. Multicard is developed by Bull and Euroclid mostly under the Esprit project Multiworks. It is commercialized by Euroclid under licence from Bull S.A.. The authors wish to thank the European Commission as well as the Bull team, Louis Salter, François Thorel and Maria Ahedo.

REFERENCES

1. Akscyn, McCracken and Yoder, 'Kms: a distributed hypermedia system for managing knowledge in organizations', *Communications of the ACM*, **31**(7), (July 1988).
2. 'Information technology—coded representation of multimedia and hypermedia information objects (MHEG)', Technical report, ISO/IEC JTC1/SC29/WG12, CD 13522-1.
3. Françoise Colaïtis and Francis Kretz, 'Coded representation of multimedia and hypermedia information objects: towards the MHEG standard', *Signal Processing: Image Communication*, **4**, 113–128, (1992).
4. Françoise Colaïtis and Francis Kretz, 'Standardizing multimedia and hypermedia objects', *IEEE Communications Magazine*, (May 1992).
5. Françoise Colaïtis and Florence Bertrand, *The MHEG standard: principles and examples of applications*, Anaheim, CA, USA.
6. A. Rizk and L. Sauter, 'Multicard; an open hypermedia system', in *Proceedings of the ACM Conference ECHT'92*, Milan, (1992).
7. F. G. Halasz and M. Schwartz, 'The dexter hypertext reference model', in *Proceedings of NIST Hypertext Standardization Workshop*, Gaithersburg, MD, 16–17 January 1990.
8. Franck Halasz and Mayer Schwartz, 'The dexter hypertext reference model', in *NIST Hypertext Standardization Workshop*, Gaithersburg, MD, January, 1990.

9. B. Amman and M. Scholl, 'Gram: a graph data model and query language', in *Proceedings of the ACM Conference ECHT'92*, Milan, (1992).
10. V. Christophides and B. Amman, Providing persistence to the hypermedia system multicard by using the oodbms o2. To appear.
11. 'F740 description générale d'un service avi', Technical report.
12. 'T170 description générale des systèmes communicants pour les avis', Technical report.
13. *Esprit Project Multiworks. Specification of the distributed implementation of Multicard on the distributed Multiworks platform.*