Reviewing two Multimedia Presentation (quasi-) Standards

Peter Rösch
Fraunhofer Institute for
Experimental Software Engineering (IESE)
Sauerwiesen-Straße 6,
D-67661 Kaiserslautern, Germany
roesch@informatik.uni-kl.de

Michael Baentsch
Department of Computer Science
University of Kaiserslautern,
P.O. Box 3049,
D-67653 Kaiserslautern, Germany
baentsch@informatik.uni-kl.de

Abstract

In this paper, we compare the BERKOM globally accessible services project (GLASS) with the well-known World-Wide Web with respect to the ease of development, realization, and distribution of multimedia presentations. This comparison is based on the experiences we gained when implementing a gateway between GLASS and the World-Wide Web. Since both systems are shown to have obvious weaknesses, we are concluding this paper with a presentation of a better way to multimedia document engineering and distribution. This concept is based on a well-accepted approach to function-shipping in the Internet: the Java language, permitting for example a smooth integration of GLASS´ MHEG objects and WWW HTML pages within one common environment.

Keywords

World-Wide Web, HTML, Distributed Multimedia Applications, MHEG, Java

1. Introduction

As more and more users are connected to computer networks, the availability of systems for the presentation of multimedia applications rapidly increases. The range of multimedia documents exchanged over networks covers simple text documents, audio or video sequences and more advanced applications like multimedia kiosks, interactive TV or video on demand. Therefore, frameworks for storage, distribution and presentation of all kinds of interrelated multimedia information become a necessity. In order to exchange and present multimedia information in resp. on different networks and platforms, standardized interchange formats must be used. Standardization is considered a key factor for commercial success of multimedia applications.

In contrast to formats for single content types like images (e.g. CompuServe's Graphics Interchange Format *GIF*), video (e.g. the Motion Picture Experts Group's *MPEG* standard format), or audio (e.g. MPEG audio, *MPA*), exchange formats for multimedia *applications* or scenarios, which include description of behavior, are still under development. Nevertheless, there are two already relatively advanced exchange formats and supporting frameworks for multimedia applications which have to be discussed in this context. The first is the de-facto standard of multimedia-presentation on the Internet, the World-Wide Web [3], which supports the Hypertext Markup Language HTML [5] document exchange format. The second framework, the so-called GLASS (GLobally Accessible ServiceS) system [13], is a prototypical implementation realizing the forthcoming commercially interesting MHEG (Multimedia Hypermedia Experts Group) ISO standard [14].

Against this background, the need for a clear and structured comparison of the two systems, World-Wide Web and GLASS, in general, and their document exchange infrastructures in particular, becomes evident. This comparison is based on the practical experience we gained when implementing the so-called *GlassWWWay* [2]. The GlassWWWay is a gateway giving users of World-Wide Web clients access to MHEG presentations. It consists of WWW servers handling requests for MHEG objects. These objects are transformed into HTML pages on the fly, i.e., the very moment a user requests them via a standard WWW browser. While implementing the necessary transformation procedures, many differences between the World-Wide Web and GLASS were surfaced.

The remainder of this paper is structured as follows. In the next section, we will describe a small sample scenario, which will be used throughout this paper when pointing out to weaknesses and strengths of both HTML and MHEG. After that, we are giving a basic introduction to the World-Wide Web and GLASS, including their respective interchange formats and communication protocols. We continue with a summary of our comparison between the two systems in section 4. As this reveals many problems with both systems, favoring one over the other does not seem to be sensible. Therefore, in section 5, we present a resolution, which -based on SunSoft´s Java programming language- integrates both systems and abolishes most of the disadvantages pointed out. The conclusion sums up the main differences between WWW and GLASS and gives an outlook on future projects.

Appeared in: Proceedings International Workshop on Multimedia Software Development MMSD-96, Berlin, April 1996

2. Sample Scenario

Our sample scenario consists of a small tourist information system. It is intended as an introduction for visitors of the city of Berlin, in which ICSE'95 is held. The information is presented using text, images and audio. In this respect, it can be part of either a typical commercial multimedia kiosk-like presentation as found for example at airports or within a globally accessible tourist-information system. In the first case, the typical execution environment will be a dedicated, stand-alone system, equipped for example with display and touch-screen. The latter might be accessible via TV or PC, offering a wide range of interaction with the presentation: From (mouse-)cursor-based selection of scenarios to cross-reference searches within online databases many sensible uses of the multimedia data become possible.

The start page shows a picture of Berlin and two buttons („Slide Show" and „Information") placed inside the picture (see figure 1). The buttons are transparent and change their state to non-transparent, if the mouse-cursor is moved inside the buttons in order to give visible user feedback. After activating the „Information"-button, a text block replaces the image. The text wraps up important informations about Berlin. Activating the „Slide Show"-button on either page will start the so-called slide show, which presents different pictures of interesting places within Berlin with appropriate textual and spoken explanations to each picture. Each picture remains on the screen for a certain period of time before it is automatically replaced by the next one. The slide show and the audio stop and the system again displays the start page, if either all slides have been displayed or the user clicks on any slide.

3. Frameworks and formats for interchange of multimedia applications

Many systems for multimedia applications exist for different domains. In our paper, we focus on systems, that:

- support applications like multimedia kiosks, on-line information systems, tele-shopping, etc.;
- feature a distributed approach, taking into account that information can be located on any server, distributed via any number of hosts, and displayed on any site within the underlying network;
- provide a well-defined and platform independent document exchange format;
- are in widespread use (quasi-standard) or are being standardized and available on all major platforms;
- exchange information about *behavior* of applications.

We have excluded many on-line systems that use a simple event-based mechanism to interact with users, because our notion of behavior demands for a higher level of abstraction to document creation and exchange. This is because we think that the provision of higher-level services is essential for the realization of complex multimedia scenarios just as it is for the implementation of complex applications. Therefore, we decided to select two of the most interesting systems matching the requirements for future multimedia systems as set forth above: The World-Wide Web and GLASS. We introduce both in the next subsections using the sample scenario of section 2.

3.1. World-Wide Web

The World-Wide Web is currently the multimedia presentation and dissemination framework used by most people. This success of the Web has various reasons like its use of the commonly available and accessible Internet infrastructure via the TCP/IP protocol stack [20], upon which the WWW´s Hypertext Transfer Protocol (*HTTP* [4]) is based. The free availability of clients (browsers) and servers (so-called *httpd´*s) alike also gave the Web its initial boost. The most important reason for the success of the Web however, is arguably its simple-to-author Hypertext Markup Language (*HTML* [5]). It permits anybody able to use a simple text editor to provide appealing content for the world, thus attracting even more people to the Web (who in turn create even more content).

As the Web is being used more and more intensively for complex and communicationally expensive multimedia documents by an ever increasing number of people, most

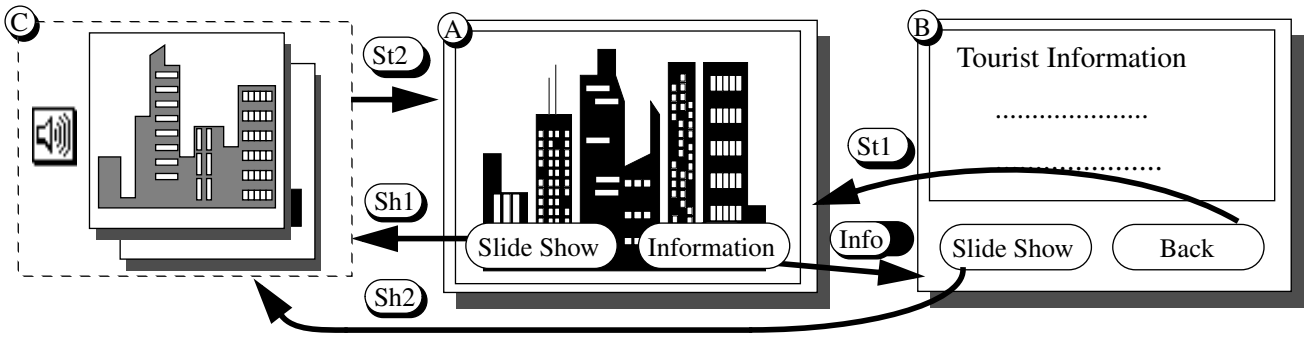

Figure 1. Logical structure of sample scenario

Appeared in: Proceedings International Workshop on Multimedia Software Development MMSD-96, Berlin, April 1996

shortcomings of the design of its *Hypertext Transfer Protocol (HTTP)* become evident. The basic reason for this is the typical communications interaction pattern used by WWW browsers: The initiation of many short, possibly long-distance TCP/IP connections overload the network bottlenecks (e.g. the internetwork-backbone routers) and the data servers alike. The detection of this behavior led to the design of the next-generation HTTP protocol, *HTTPng* [19], which has been formally designed using ASN.1. However, it will again be specialized towards transmission of HTML documents and will thus be not generic enough to take topics like quality of service into account, although this ought to be of special importance in most multimedia presentations.

The *Hypertext Markup Language (HTML)* has been derived from the Standardized General Markup Language (SGML [21]) for mainly non-interactive presentation of primarily hypertext with embedded pictures like diagrams for the explanation of technical content. This relatively simple hypermedia approach was not meant to be used to realize today's highly interactive multimedia presentations for which our small scenario is a simple example. We will present the many shortcomings of HTML in this context while explaining how we realized the given scenario with the techniques developed during the implementation of the aforementioned GlassWWWay. A very basic understanding of the features of HTML is necessary to completely follow the ensuing discussions of an implementation of our sample scenario within the Web. Since such explanations have become abundant in the meantime and since this would clearly be beyond the scope of this paper, we only refer the interested reader to for example [1].

In order to realize the chosen scenario, we first show how to divide the problem into manageable parts. The first step in this development process for the page-oriented HTML language is to identify the different documents that have to be created. In this case we see 2+N pages with N equaling the number of slides in the slideshow to be appropriate. The first document consists of a GIF image showing the start picture of the scenario (A), the second contains hypertext with the information about Berlin (B), and finally, one page for each single picture of the slideshow.

In order to obtain the desired look and feel, we had to use several non-standard approaches on each page to make it work as desired. Page A for example consists really of three pictures merged with a tool into one. The three pictures are the original background and the two buttons. For purposes of this sample scenario, the creation of this combined picture has been done manually, but it could also take place on-line, e.g., as the page is being prepared for transmission. This is exactly the strategy chosen for the realization of the GlassWWWay, but its drawback is obvious as many people use this gateway: The solution does not scale: As many clients access a server providing such a computationally expensive operation, the host of such a server quickly bogs down under the load. As opposed to the complex structure for page A outlined above, the page associated with state B simply consists of the information hypertext coded in HTML, since this is the most natural way to present it given the medium WWW browser. We only added code for two simple hyperlinked GIFs displaying the two buttons associated with the next state (slideshow or start page) since we did not had to render the text into a picture given the hypertext-presentation capabilities of Web-Browsers. Finally, the pages making up the slideshow are realized as HTML documents consisting basically of two objects. The first is the image to be displayed and the second is a hyperlink to the audio file associated with the picture. This detour, where the user has to explicitly request the playing of the audio file is necessary since the currently available and standard Web mechanisms do not permit the transmission and presentation of different types of multimedia objects at the same time. Another topic is that the ability to play back audio is usually not build into WWW browsers, probably because too many different audio formats are in use. The only viable way here is to use client-external programs spawned in response to incoming data of a certain MIME [7] type (e.g. MPEG-audio) completely bypassing the browser.

After the different pages had been designed, they had to be interconnected to produce the desired presentation, i.e., the state-transitions as shown in section 2 had to be realized. We had to use three different techniques to obtain the desired functionality. For page A, a specialized WWW server-side executable, a so-called *cgi-bin* (i.e., an executable conforming to the Common Gateway Interface [15] on the server side) had to be installed. After configuring it to process the information of the mouse cursor position over the composed GIF on the browser making use of the ISMAP-tag of HTML, it determined which page to present next. The slideshow itself, however forced us to use a rather dirty trick, i.e., a currently still non-portable and client-specific technique, because the interaction pattern as outlined in our scenario was not foreseen by the design of the Web. We believe our realization to be of use anyway, since we are able to make the two most widespread browsers, NCSA Mosaic and Netscape's Mozilla (in use by about 90% of all people accessing the Web) accept it. We use the Common Client Interface (*CCI* [18]), the client side equivalent to the server side CGI, to force the loading of the different pages making up the slideshow by the clients. Thus, every few seconds, as indicated in the slideshow specification, a command is sent from a server-side executable activated by traversal of the hyperlink associated with transition (Sh1) or (Sh2) to the clients, thus indicating which page of the slideshow to load next. Finally, all images of the slideshow have been marked up with hyperlinks whose traversal causes the server side executable to stop issuing slide-reload requests and return to the initial page as required by transition (St2).

As it should have become clear during the presentation of the implementation of this simple scenario, the WWW´s

HTML Language is clearly not suited for this kind of multimedia information presentation. If the various trick and tools briefly described above were not available, our implementation of the GlassWWWay would have been impossible. This clearly underlines the unsuitedness of the abstractions available in the current Web infrastructure for the creation of multimedia presentations.

3.2. GLASS

The *GLASS* project [13] aims to realize globally accessible multimedia services in a standardized and in this respect open way. The services addressed by GLASS are oriented towards commercial utilization. They include consumer services like tele-shopping, Pay-Per-View, Video-On-Demand, on-line-catalogues, and more. The project has been initiated by the DeTeBerkom and several partners worked together under its supervision. Partners included both industry (e.g. Digital, IBM and Grundig MM) and academic research institutions (e.g. GMD, University of Berlin and University of Kaiserslautern). The prototypical system which has been implemented in the GLASS project has already been presented successfully (e.g. at the ISS'95, IFA'95 and Telekom'95) and the client systems are supported on all major platforms (e.g. Intel-PC, Sun-Sparc, Digital-Alpha and Apple-PowerPC). The components developed within GLASS include user interface agents, content and accounting servers, gateways to other services and authoring tools.

All data exchanged between these components is encoded using the *MHEG* standard. Within GLASS, a textual representation of MHEG objects, called *MDL* (MHEG Description Language) was developed. An MDL compiler encodes MDL files to MHEG objects using ASN.1. The GLASS project does not only use MHEG, GLASS partners are also involved in the standardization process. Therefore, early feedback can be provided to the MHEG group.

The future MHEG standard (see table 1 for a listing of MHEG subsets) is designed to meet the requirements of multimedia applications and services, running on heterogeneous workstations interchanging information in real-time. The envisioned applications and services are for example systems for computer-supported cooperative work, multimedia messaging, audiovisual telematic support for training and education, simulation and games, Video-On-Demand, interactive TV-guides, and others [9].

MHEG is being standardized by the ISO/IEC (JTC1/SC29 WG12). MHEG subsets 1,2 and 3 have reached the Draft International Standard (DIS 13522) state. MHEG-5 is yet a Committee Draft and MHEG-4 is likely to disappear from this list soon.

Features of MHEG include specific interaction structures for real-time interchange of multimedia data, composition and synchronization of multimedia data in space and time, linking between elements of composite multimedia objects and reuse of multimedia data in different contexts. Applications encoded using this forthcoming standard are „end-formatted". This means, that each application can define its own „look and feel". The standard itself does not propose a typical graphical user interface.

1	MHEG Object Representation (ASN.1)
2	Alternate notation (SGML)
3	MHEG Extensions for Scripting Language Support
4	Registration Procedure for Format Identifiers
5	MHEG Subset for Base Level Implementation

Table 1. MHEG levels

The class hierarchy shown in figure 2 is fixed and cannot be extended by an application. Nevertheless, once an object has been decoded within an MHEG engine, it can be reused when creating other objects. Composite and content objects are referenced when creating the so-called *runtime objects*. The runtime objects remain at the presentation system and they are not exchanged. Speaking in the terminology of user interface technology [17], exchanged MHEG objects are *models* and runtime objects are *views*.

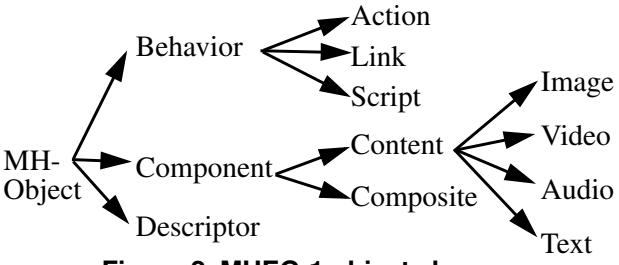

Figure 2. MHEG-1 object classes

We have encoded the scenario described in section 2 applying MHEG-1 based tools of the GLASS project. Authoring has been done using a textual editor and the GLASS specific MDL language. In the following subsection, we will present a small extract of our encoding.

The starting page of our scenario (A) expressed in MHEG-1 consists of several „model"-objects which are contained in a single *composite object*. Composite objects are used for structuring, for associating multimedia and hypermedia objects and to define a set of objects that can be downloaded as a whole instead of as single parts. In the case of our scenario, we have put all information about the start page (A) and the information page (B) into one single composite object. This composite object contains all *content objects* (e.g. references to JPEG files) needed for the start page and the information page.

As an example for MDL-code, the description of the link `Create_slides_rt_obj` is listed on the top of next page.

The *link class* in MHEG is slightly different from what you would expect if you are familiar with the term link known from hypertext [10]. Link objects express e.g. inter-

```
link Create_slides_rt_obj {
   trigger { is-prepared Slides_btn_rt_obj },
   action-list { parallel,
      action-list { serial,
         action-list { target Sbtn_normal,
            action { type new, template Sbtn_normal_rt_obj }
      }
      action-list { parallel, target Sbtn_normal_rt_obj,
       action { type set-attachment-point, position-x 50, position-y 550, position-z 1},
       action { type set-selection-style,selection-style button }
     } }
     -- same for Sbtn_hlight_rt_obj and Sbtn_pressed_rt_obj
  } }
```

active behavior in a multimedia presentation. They consist of trigger conditions and action objects. The trigger condition can be described using simple logical operations and attribute or status requests applied to an object. When the trigger condition becomes true, the action object is executed. Therefore, a link object defines an n:m relationship between objects referenced within the trigger condition and target objects referenced within the action object. The link in our example creates new runtime objects for the states (normal, highlighted and pressed) of the „Slide Show"-button and sets the position and selection-style of each created button-image object.

Creating the presentation elements does not mean that all three images are visible at the same time. The visibility of image objects is controlled by the actions „run" and „stop". Links are needed for all transitions of each button element (normal to highlighted, highlighted to normal, and highlighted to pressed). The link specifying the transition from highlighted to pressed contains additional actions in order to enter another state of the scenario (e.g. Info).

Summing up, our starting composite consists of the following elements:

- Model objects: page backgrounds (e.g. `Start_background`) and button composites (e.g. `Slides_btn`).
- Preparation link: a link, which readies all model objects. All the content data for the images is then retrieved from the content server.
- Links for creation of runtime objects (e.g. `Create_slides_rt_obj`). These links are triggered automatically after the model objects have been prepared.
- Transition links for specification of button behavior (e.g. `transition_normal_to_highlighted`).
- Transition links to other pages (e.g. Sh1, Sh2).
- Start-up link: performs a „run"-action on background and button objects of the first page and thus makes them visible.

The transition to the „Slide Show" page (Sh1 and Sh2) is performed by an action-list that stops all currently active runtime objects and prepares the new composite `Slide_one`. Similar to our start page, first the slide-picture and the audio objects are prepared. When the run-action is activated, the image is displayed and an audio-stream is transmitted from server to client. Additionally, a so-called timestone is activated. The next slide is presented if the timestone expires. If the user clicks onto the picture, a transition to the start page is triggered. The rest of slides is encoded similar to `Slide_one`. Only in the case of the last slide, the timestone triggers a transition to the start page instead of another slide.

SMCP	Session Management Control Protocol
AUTH	Authentication Protocol
CRED	Credit Control Protocol
SCP	Store Control Protocol
POCP	Presentation Object Control Protocol
PODP	Presentation Object Data Protocol

Table 2. GLASS protocols

The *GLASS protocols* define the procedures for the exchange of data between all agents that exist in the system (see figure 3). In GLASS, agents can be presentation systems at the client side, content servers, or accounting/security agents. Depending on the type of the two communication instances, the GLASS protocols listed in table 2 have been defined.

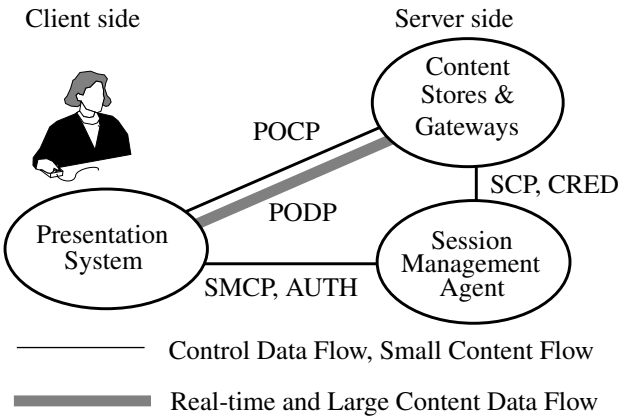

Figure 3. GLASS agents and protocols

Communication links between clients and servers are

stateful. Thus, presentations can be driven by data streams instead of down-loaded files. Furthermore, clients and servers can have multiple open communication links at the same time. The communication links and the transmission of content data is controlled by the POCP protocol. The PODP protocol defines how to transmit the content data (i.e., a large data flow) itself. PODP is a one-way protocol from content servers to the client site.

Security and accounting is controlled by a session management agent. It tells the content store to transmit content data to the presentation system (using the SCP protocol) after the appropriate authentication and credit messages have been received from the client (AUTH and CRED protocol).

4. Comparison

When the need for a decision between an MHEG or HTML-based system for a multimedia task arises, the respective application and its context must be carefully taken into account. This comparison should help multimedia designers to get an overview about advantages and disadvantages of the systems under discussion. However, everyone has to make the choice for oneself, based on the application to realize, the available resources, and the side considerations to be taken into account. As we have mentioned earlier, the experiences we made when implementing the GlassWWWay provide a basis for our comparison. Because this gateway has potentially to translate any MHEG scenario into an HTML presentation, we have experienced the limitations and advantages of both frameworks first-handedly.

In this comparison, we aim to consider all aspects deemed to be important for multimedia designers deciding between MHEG and HTML right now. Therefore, we judged only capabilities that are already available and that can be used right now. For example, since no browsers supporting scripting in MHEG and no clients supporting HTML client-side imagemaps were available at the time of writing this comparison, these experimental features are not mentioned in the summary.

The GLASS prototype has been implemented using TCP/IP as its communication protocol. Nevertheless, GLASS has been targeted at the realization of services requiring stateful connections (e.g. B-ISDN). This is the reason for some of the disadvantages of MHEG, like system-inherent bottleneck and scalability problems. Applications like Video-On-Demand require not only stateful connections but also a large amount of communication bandwidth and cannot be driven by simple request-download protocols. Therefore, problems arising because of stateful connections are common to multimedia applications like Video-On-Demand and not only MHEG-specific.

We have divided the important design aspects into four areas. *Application domains* describes the environment each system is most useful for. *Authoring/abstraction* discusses the suitability of the available authoring and display tools. The *Protocols and performance* part summarizes aspects of transmission efficiency, whereas the *standardization and availability* part highlights the availability of software. The results of this section are summarized in table 2.

4.1. Application domains

The application domains of both systems obviously overlap. However, sometimes it is tricky and not very straightforward to apply either framework to a domain it was not quite designed for. In this subsection, we only state the typical application domains of GLASS/MHEG and WWW/HTML.

The most important domain of HTML are hypertext applications. Because the World-Wide Web provides access to a distributed hypertext storage system based on HTTP servers, it is well suited for global text-based information services. HTML also permits image files and audio or video sequences as targets of hypertext links. However, real hypermedia structures, i.e., the ability to establish links between all types of documents to one another is not well provided by the Web. This is why applications like multimedia kiosks, product demos and animations are not easily supported by HTML. The level of user interaction and integration of different media is rather low.

MHEG supports highly interactive multimedia applications like Video/Audio-On-Demand, Pay-Per-View, multimedia product demos and tele-shopping much better. The encoding of documents however, is very complicated, which becomes especially obvious when trying to realize simple hypertext information systems with MHEG. However, it is possible to support such applications given the right tools.

4.2. Authoring and abstraction

With HTML, both a format for data exchange, as well as an authoring language are defined. In contrast to HTML, MHEG defines only the format for data exchange. Therefore, we are comparing MHEG's data description language MDL with HTML, since MDL provides access to all components of MHEG in a straightforward and more abstract way.

The methods used to describe applications in HTML or MDL are very different. In an MDL document, the end-format of a presentation is described, including the layout of the presentation. Using HTML, the author inserts some markup elements into the text only. Therefore, the authored document is much closer to the logical hypertext structure, than it is possible using MDL. In MDL, much more information must be included into each multimedia document to be presented because MHEG does not predefine any kind of "look-and-feel" of applications. Therefore all basic components that define an interaction must be included (e.g. compare the description of a button which was introduced in section 3.2). Of course, authoring tools can hide these low level details (as it is achieved in MDL by providing a macro package), but the MHEG objects that are generated by the authoring process still remain pretty large.

MHEG tries to support reuse by providing model and runtime objects. But, as you can see by our example in section 3.2, this concept fails as soon as the objects to be reused get complicated. For example, it is not possible to overload the normal, highlighted and pressed images in a runtime object because these images are attributes of the

model class, not of the runtime object. This also explains why MHEG-based documents that are comparable in functionality with HTML pages are usually more inefficient in terms of communication bandwidth and secondary storage used.

On the other hand, an author has the opportunity to specify any kind of behavior or look-and-feel when using MDL. Although some important elements of object-oriented languages are missing, MDL is more an object-oriented programming language, than a document description language like HTML. Therefore, MDL provides capabilities for describing highly interactive multimedia applications, as opposed to HTML. But, in order to support a higher level of abstraction in the MHEG based authoring process, other tools like e.g. WYSIWYG editors are needed. The concept of author-defined look and feel is not natural to HTML since it has been explicitly designed around the principle of freedom to render documents as each client chooses. A first step into the direction of author-induced layout however, has been made with the introduction of style sheets into the HTML 3.0 standards draft and first browser implementations.

A major drawback to using MHEG is its current lack of production-level tools, as are available for HTML. This is basically because of the huge support for HTML and the relatively low profile the still evolving MHEG standard has.

4.3. Protocols and performance

One of the striking differences between GLASS and the Web is the topic of state. While the GLASS user interface agent (UIA) and the session manager explicitly contain state-machines and are thus stateful on client- and server-side alike, the Web favors the principle of statelessness for its servers. The World-Wide Web and its protocols have been developed around a download-display-download cycle as common in the fault-tolerating (and error-prone) Internet environment. GLASS protocols are advanced in the sense that clients are enabled to handle multiple continuous media streams at the same time, incurring state at both sides of the communications link. This however underlines the lack of scalability of the systems even further: As more and more clients use them, they more (GLASS) or less (WWW) quickly cease to deliver reasonable performance for the end-user.

Nevertheless, an advantage of MHEG is that it allows to take performance aspects into account. As an example, an application can prepare all model objects before the presentation is started. If video objects are part of the application, the required bandwidth might be reserved for the time the presentation lasts via yet-to-be-defined means. Furthermore multiple MHEG objects can be grouped together and be transmitted as a whole. This transmission unit is not restricted to a single page as in HTML, but it could be for example a complete multimedia scenario like a product demonstration. Since in many computer networks, single large packages are better for data transmission performance than multiple small ones, this feature is in certain environments advantageous.

4.4. Standardization and availability

As MHEG is a forthcoming international standard, all properties of the exchange format are fixed and platform independent. The same is true for HTML, which can be considered to be the de-facto standard for distributed hypertext information systems. The availability of MHEG and HTML is quite different. World-Wide Web and HTML-based tools and services are available world-wide both from commercial and from non-profit providers. Supported by the infrastructure of the Internet, the Web is currently the most well-known, available and used hypermedia system. Another advantage of this wide distribution is the fact, that many research and commercial groups are working on extensions to the Web, trying to overcome the deficiencies of the current WWW while still being compatible to existing HTML-based applications.

The currently very low availability of MHEG is a potential risk for investment in this area. The success of MHEG also heavily depends on the availability of supporting tools, especially high quality tools for browsing and authoring of applications. Currently, only a few prototypical and not very sophisticated tools are available.

5. A possible solution

After the above comparison of the two more or less established standards under discussion, in this section, we give a brief overview about a not yet as widely available technology we believe to be the future of multimedia information presentation on the Internet and position its capabilities in the established framework of the previous section.

Java is a flexible, extensible, object oriented programming language developed by SunSoft [12]. It closely resembles in its syntax C++, but it is also deemed to be secure in the sense that no programs written in Java can negatively influence any resource like the filesystem on the machine executing Java programs. This feature makes an interpreter of Java-object code guarding all resources a necessity on any host running Java programs, so-called *applets*. With these prepositions, it becomes clear that applets are meant to be used for function-shipping between hosts with a Java runtime environment.

The ultimate rationale behind this is the aim to reduce the amount of data transmitted for any given scenario, because now behavior, i.e., functions, are shipped and no longer large data chunks like precomputed data sequences. This is sensible given the huge amount of computing horsepower available on the client-side hosts, while the servers at the same time are too often overloaded with too many client's requests.

Before explaining how to work with Java, we briefly highlight the short, but nevertheless remarkable history of this language. It was first (semi-)officially introduced in March 1995 with a posting about its availability for testing to the respective WWW-related newsgroups. From the very beginning, it contained a large number of useful classes, among which a complete WWW browser, *HotJava*, demonstrated the usefulness and simplicity of the concept. This initial distribution therefore contained already classes

for HTML parsing, all major Internet protocols, like HTTP or FTP, and a plethora of helpful tool classes, e.g. general-purpose GUI classes or mathematical functions. Its first major boost, however, Java got from its presentation at the third International World-Wide Web Conference in Darmstadt, Germany in April 1995. The official presentation of a stable alpha-release took place at the end of May '95 at SunWorld in San Jose, California. During the summer of the same year, a Java programming contest helped spur even more interest in the language and helped create even more applets than were already available with the official Sun release. The last and most important event was the introduction of a Java runtime environment into Netscape's Navigator, the most widely used WWW browser currently available. With this, Java applets can all of a sudden be run on roughly 80% of all hosts connected to the Internet, a capability it owes to its machine-independent bytecode, that only has to be interpreted by an appropriate runtime. This integration of Java with Netscape caused the introduction of a slightly modified API (application programmer's interface), the so-called *Beta-API*, which can be seen as the first step into the direction of standardization of Java.

The next interesting question to be answered is, how to work with Java, how to develop Java applets? One has to learn a new programming language, but the support is rather good: From day one of public accessibility, the Java development environment contained for example a (rather slow, but correct) working compiler for transforming Java code into the machine independent bytecode of the Java runtime. Thus, program development as usual for compiled languages is possible, while the created bytecode could still be checked if the need arises. This is why anybody can develop applets providing some behavior hitherto not available, compile it into code executable by the low-level Java interpreter, and place this code on the Web in order to let anybody access this new functionality. Since a new markup tag has been introduced into HTML for this purpose, Java programs can be directly blended into the Web, allowing a dynamic and secure enhancement of browser behavior 'on-the-fly'.

The language itself resembles in most structures the object-oriented language C++, but without the problematic features of C++. For example, Java features garbage collection of dynamically allocated data, thus alleviating programmers of the burden of memory allocation problems. It also knows the concept of threads as the units of execution, which extremely simplifies the development of programs with several threads of control. From the standpoint of program development, this is extremely sensible in inherently concurrent environments, like user interface and/or network protocol programming, see e.g. [8]. The last distinguishing feature to be mentioned here is the strict type concept, which cannot be weakened by casting as possible in C++. This, together with the trusted runtime, which does for example array-bounds checking, guarantees the safeness of Java code in the sense that no "foreign", i.e., downloaded, applet can negatively influence any resource on the client's host, a guarantee no object code generated by a foreign (C++) compiler can give.

Besides these advantages of the language itself, further positive topics are related with the relative youth of Java. One is for example, that advanced software engineering practices have been applied right from the start: On the one hand side, the language itself features a clean, no-nonsense design, where no backwards-compatibility compromises had to be made. Secondly, the availability of a complete repository of Java applets for any given purpose [11] makes the search for and the reuse of any given Java software unprecedentedly simple.

Another advantage of Java in the context of multimedia document development and presentation is the raised level of abstraction: Anybody can make use of existing applets as easily as writing HTML code has been before. The power of expressiveness is as high as the abilities of the respective Java applets permit. As an example for this statement take the following code excerpt from our Java implementation of the scenario we used to compare HTML and MHEG:

```
  <app class="StoppableAnimator"
img="Berlin" back=Start.html pause=1000
   order="1:200@audio/pic1.au|2@audio/
pic2.au|3:@audio/pic3.au"
   repeat=true height=900 width=550>
```

It realizes a continuously running (3-)slide show with integrated audio presentation without the need to use some non-standard HTML/WWW 'tricks' (see section 3.1) or to author complex interactions (see section 3.2). `StoppableAnimator` designates the Java class to be downloaded into the Java-enabled browser to display the sequence of images in the subdirectory '`Berlin`' of the same WWW server at which the scenario is located. Of course it was just plain luck to already find an applet having nearly the functionality we needed for our scenario, but the probability this will happen on any given problem will rise as more and more people write applets and make the respective object code (not necessarily the source) accessible via the WWW.

Since another applet providing the client side imagemap behavior of state A in the scenario of section 2 was also readily available, the overall coding effort that went into the realization of our sample scenario was roughly two orders of magnitude lower than what we had to put into the respective realizations using (plain) HTML or MHEG. This is even more surprising as we had to slightly modify the original applet `Animator` to accept another resource, namely the `back` attribute indicating which page to load if a user stops the slide show by pressing any button.

The last point to be made here is that the division of writing Java-enhanced HTML code on the one side and new Java applets on the other side is most certainly sensible from what we experienced when developing any Web-application, be it WebMake [6], the GlassWWWay, or the simple multimedia scenario outlined in section 2: Most of the time, one can get along with high-level and in terms of ease of understanding, simple, document presentation abstractions. As soon as unforeseen behavior has to be integrated, though, major efforts in terms of using WWW 'tricks' or complex MHEG scenario development have to be undertaken and are as such only seldom reusable at another project. As opposed to that, Java's framework of gen-

Continuous media (e.g. Video-On-Demand)	very well suited	very limited	well suited
Determination of look-and-feel	authoring time	rendering time	any time
Description of behavior	yes	no	yes
Abstraction level	low	high	very high
Ease-of-use	complex code	simple	simple / difficult
Code compactness	low	high	high
Units of transmission	scenes	pages	pages+functions
State / -> Degree of scalability	client & server side / Low	client side / Medium	client side / High
Bottleneck	server & network	network	network, (client host)
Standardized	international	de-facto	inter-vendor β-API
Availability of software	few prototypes	broadly available	many platforms
Accessibility / Market penetration	few projects only	world-wide	rapidly rising

Table 3. Overview about the important features of the surveyed systems

6. Conclusion

In this paper we pointed out the areas of application for two (quasi-)standard multimedia document exchange formats, the Hypertext Markup Language of the World-Wide Web and the Multimedia and Hypermedia Expert's Group MHEG format. We implemented a scenario typically found in today's multimedia presentations with both approaches. Based on this and several purpose object-oriented classes which can be reused in any context once they have been written clearly has a leading edge in terms of economics of multimedia document engineering.

In the last part of this section we summarize the features of Java and HotJava in the same table format we used for comparing HTML and MHEG (cf. section 4). Since most entries are obvious or self-explanatory after the presentation of Java's capabilities above, we only concentrate in this concluding part of the section on Java on a selected subset of features of the combination Java/HTML.

The first point we would like to stress is the topic of scalability. Java applets are inherently the carriers of state, and since they are located on the client side, this approach scales with the number of clients as long as the clients, that may thus be the only bottlenecks, are computationally powerful enough. As opposed to that, any approach storing state on the server side is doomed as soon as a rising number of clients accesses the respective documents. Of course, there exist certain applications (e.g. Video-On-Demand) that make a state necessary in any case.

As ease of use is discussed, one has to carefully distinguish between users and programmers of Java. While the first only have to know how to integrate existing applets into HTML pages (which indeed is trivial), the latter have to be able to master the new programming language. This directly leads to a very high level of abstraction for the users of the combination Java/HTML. Nevertheless, programmers even get the opportunity to modify the mechanisms and the look-and-feel of a Java-enabled display agent like a WWW browser.

The last point to be highlighted here is that of availability of Java and more importantly, its runtime environment. Since the most influential WWW software provider, Netscape Inc. licensed Java and ported it to all major platforms, the system emerged from its niche of Sun Workstations running Solaris. The additional acceptance of Java by both Microsoft and IBM, which both licensed the technology as well, shed a promising light on the language, its concept, and overall availability in the future.

After all, the lesson we learned is to use Java as the foundation for a next-generation gateway between MHEG and WWW, a work currently under way.

Aspect to be compared	MHEG	HTML	HTML+Java
Hypertext information systems	not well suited	very well suited	very well suited
On-line multimedia kiosk	very well suited	not well suited	very well suited
Interactive multimedia applications	very well suited	very limited	very well suited

Table 3. Overview about the important features of the surveyed systems

lier experiences when conducting WWW and MHEG projects, we extracted the highlights and weak spots of both frameworks.

The summary drawn is that MHEG is a high risk in terms of investment in a standard that might be coming too late. Nevertheless, we are seeing one possible market for MHEG which is Video-On-Demand and interactive-TV because here connections are explicitly reserved for each client (and which are charged). In contrast, HTML established itself quite firmly due to its large support base, the many tools available, and its free accessibility, although it is not a real standard and was merely meant for hypertext documents.

We concluded the paper with the presentation of Java, and how it smoothly blends into the existing infrastructure of the Web, extending its possibilities and deleting most of the drawbacks pointed out in our evaluation of HTML alone. Since MHEG engines can be built using Java technology in general, and Java classes in particular can be defined for MHEG, the use of Java as a base technology for multimedia document development, exchange, and presentation on the Internet seems to be the most sensible way to go in terms of both economic and engineering considerations.

References

[1] Andreesen, M.: *A Beginner's Guide to HTML*; URL = http://www.ncsa.uiuc.edu/demoweb/html-primer.html, 1993.

[2] Baentsch, M., Rösch, P.: *Weaving interactive media into the Web: The WWW-GLASS gateway*; Proceedings Workshop on Interactive and Distributed Multi-Media Systems on Highspeed Networks at the 3rd International World-Wide Web Conference, Darmstadt, Germany, April 1995, pp. 4-8.

[3] Berners-Lee, T., et.al.: The World-Wide Web; Communications of the ACM, August 1994, Vol. 37, No. 8, pp. 76-82.

[4] Berners-Lee, T., Fielding, R.T., Nielsen, H.F.: HTTP/1.0 Internet Draft 03, Best Current Practice; Internet draft; URL = http://www.w3.org/pub/WWW/Protocols/HTTP1.0/draft-ietf-http-spec.html, 1995.

[5] Berners-Lee, T., Connolly, D.: *Hypertext Markup Language - 2.0,* Internet draft; URL = http://www.w3.org/pub/WWW/MarkUp/html-spec/html-spec_toc.html, September 1995.

[6] Baentsch, M., Molter, G., Sturm, P.: *Booster: A WWW-based prototype of the Global Software Highway*; Proceedings 2nd International Workshop on Services in Distributed and Networked Environments; Whistler, Canada, June 1995, pp. 156-165.

[7] Borenstein, N., Freed, N.: *MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies*; Internet RFC 1521, September 1993; URL = http://www.oac.uci.edu/indiv/ehood/MIME/1521/rfc1521ToC.html.

[8] Buhler, P.: Dissertation at the Computer Science Department of the University of Kaiserslautern, Germany, June 1993.

[9] Colaitis, F.: *Opening Up Multimedia Object Exchange with MHEG*. IEEE Multimedia, pp. 80-84, Summer 1994.

[10] Conklin, J.: *Hypertext: An Introduction and Survey*. IEEE Computer, Sept. 1987.

[11] Gamelan: A repository for Java applets; http://www.gamelan.com/Gamelan.html

[12] Gosling, J., McGilton, H.: *The Java Language Environment: A White Paper*; URL = http://java.sun.com/whitePaper/javawhitepaper_1.html, 1995.

[13] Hofrichter, K.: *Berkom GLASS (GLobally Accessible ServiceS)* (project overview); URL = http://www.fokus.gmd.de/ovma/berglass/entry.html.

[14] ISO/IEC CD 13552: *Information Technology - Coding of Multimedia and Hypermedia Information Part 1 - 5*, 1995.

[15] McCool, R.: *The Common Gateway Interface*; URL = http://hoohoo.ncsa.uiuc.edu/cgi/, 1994.

[16] Meyer-Boudnik, T., Effelsberg, W. *MHEG Explained*. IEEE Multimedia, pp. 26-38, Spring 1995.

[17] Myers, B.A.: *User Interface Software Tools*. ACM Transactions on Computer Human Interaction; Vol. 2, No. 1, March 1995.

[18] National Center for Supercomputing Applications: *NCSA Mosaic Common Client Interface*; URL = http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/CCI/cci-spec.html, April 1995.

[19] Spero, S.: *Progress on HTTP-NG*; URL = http://www.w3.org/pub/WWW/Protocols/HTTP-NG/http-ng-status.html, 1995.

[20] Stevens, W.R.: *TCP/IP Illustrated*, Vol.1, Addison-Wesley, Computing Series, 1994.

[21] Cover, R. Duncan, N. Barnard, D.: *The Progress of SGML (Standard Generalized Markup Language): Extracts from a Comprehensive Bibliography*. Literary and Linguistic Computing 6/3, pp. 200-212, 1991.