Meta-data Modelling for Quality of Service (QoS) Management in the World Wide Web (WWW)

E. Madja[1], A. Hafid[2], R. Dssouli[1], G.v. Bochmann[1], and J. Gecsei[1]

[1]Université de Montréal, Dept. Informatique et Recherche Operationelle,
Montréal, Canada; E-mail: {madja, dssouli, bochmann, gecsei}@iro.umontreal.ca

[2]Advanced Communication Engineering Centre, University of Western Ontario,
London, Ontario, Canada, N6A 5B9; e-mail: hakim@csd.uwo.ca

Abstract

The World-Wide Web has been a remarkably successful system for distributing hypertext documents. The basic model for Web interaction is that a client requests a page of data which can include images and hyperlinks within it. This interaction model is inadequate for real-time multimedia (MM) applications, since the Web and its associated set of protocols, e.g. HTTP, do not support the real-time transfer of the continuous media (Audio/Video). Several solutions have been proposed to support real-time playout of continuous media via the Web, e.g. Netscape. Most of these solutions do not provide means to the user to negotiate the desired presentation quality (in terms of quality of service (QoS) parameters settings); even the proposals that provide QoS negotiation (more generally QoS management) are used in a rather static manner, that is, the video/audio servers are a priori known. In this paper we propose to integrate in the WWW a dynamic QoS management approach that allows (1) the user to negotiate the desired QoS; and (2) and to select the "best" video/audio server which might support the user requirements. This activity is based on the general structure of multimedia documents and associated QoS parameters, we call meta-data, which we developed under an ongoing CITR project. The main objective of the paper is to integrate meta-data associated with MM document in WWW, e.g. Netscape; this will allow us to use our dynamic QoS management protocols.

1. Introduction

In the context of a CITR (Canadian Institute for Telecommunications Research) project, a prototype system for remote access to News-on-Demand [4, 17] has been developed. The prototype provides facilities to store, retrieve and present multimedia (MM) news information. The prototype is an integration of software components developed by the various teams working on the different sub-projects components of the major project. The prototype consists of the distributed multimedia database (DBMS) from University of Alberta [15], the distributed continuous media file (CMFS) server from University of British Columbia (UBC) [11], the synchronization component from University of Ottawa [9], a scalable video decoder from INRS [2], and the QoS management component from University of Montreal [5].

With the current prototype, the user retrieves through a database query interface one or more news documents. For example, the user could ask for the most recent news on *Stanley Cup*. Then, he/she selects one document (article) to be played and sets the desired QoS, which includes the desired video resolution, video color, etc. The QoS negotiation features of the prototype allow to find an optimal configuration of system components which might support the delivery of the document while satisfying the user requirements. This activity supports database distribution, and uses a MM document model where the monomedia components of the document may exist in several variants. The variants may support different QoS and may be stored at different servers [1]; for example, the same video sequence may exist as a MPEG2 file and also as MJPEG file which are located at different servers. The negotiation operation (performed by an entity, we call QoS manager) requires interactions with the different system components: with the DBMS to get meta-information on the document; with the CMFS to evaluate its capacity to deliver MM data; and with the synchronization component to evaluate the synchronization quality to present the document. If the QoS negotiation is successful, then the document may be presented right away. However, if the negotiation failed then the best available offer (which does not, however, satisfy the initial user requirements) is presented to the user who may accept the offer, initiate a new negotiation, or abandon

the session.

In this paper we aim to integrate the ideas behind the CITR prototype in terms of QoS management in the WWW; to show the practicability of this integration, we produced a version of the CITR prototype compatible with the Web (specifically Netscape). Basically, the idea is to be able to choose between different media variants from different sites depending on the user specified preferences and the system capabilities. The project is motivated by the fact that the Web and its associated set of protocols do not support adequately the real-time transfer of the continuous media (Audio/Video). This inadequacy is due to the fact that the underlying Internet transport protocol (TCP/IP) does not perform resource reservation for real-time data traffic. Sometimes, Audio/Video clips data are completely downloaded before being played back by an appropriate application. This scheme becomes easily impractical for long video clips. Our project foresee access, through the Web, to multimedia documents that are distributed over heterogeneous servers like the http servers (for text and image components) and the Continuous Media file servers (CMFS) that was used in our initial prototype. Other servers such as Vosaic and RealAudio/RealVideo will also be considered for providing components of a multimedia document. The Web-compatible prototype provides a user interface for the QoS negotiation which enables the user to specify his/her preferred media presentation quality. The Audio/Video component can be included in the HTML pages as a Java applet, Helper application, or a plug-in (a button on which the user clicks to actually access the audio/video component). When a user clicks on the button, a graphical user interface appears that lets him/her specify his/her preferences. These preferences and the accessibility of the different media variants are analysed first, and then an appropriate variant is presented to him. To be able to support the Quality of Service (QoS) function on the Web, it is necessary to have access to meta-data of the multimedia documents to play. Two solutions are identified: (1) store the meta-data locally (or in a specific meta-data server) in a database or in simple files, e.g. Unix files; or (2) define an extension for the HTML language conveying the media meta-data related to QoS issues.

The rest of the paper is organized as follows. Section 2 describes the meta-data required for our QoS management approach. Section 3 describes the integration of the meta-data of MM documents in WWW. Section 4 discusses the implementation issues of this integration and presents an implementation in the context of our CITR prototype. Finally, Section 5 concludes the paper.

2. Multimedia document and meta-data

A MM document could be multimedia or monomedia. A multimedia document is composed of several monomedia. Figure 1 shows the structure of a MM document using the notation of OMT object model [13]. It shows that a document is either a monomedia or a multimedia, and that a multimedia is composed of one or more monomedia (aggregation links), and has attributes which consists of spatial and temporal synchronization constraints.

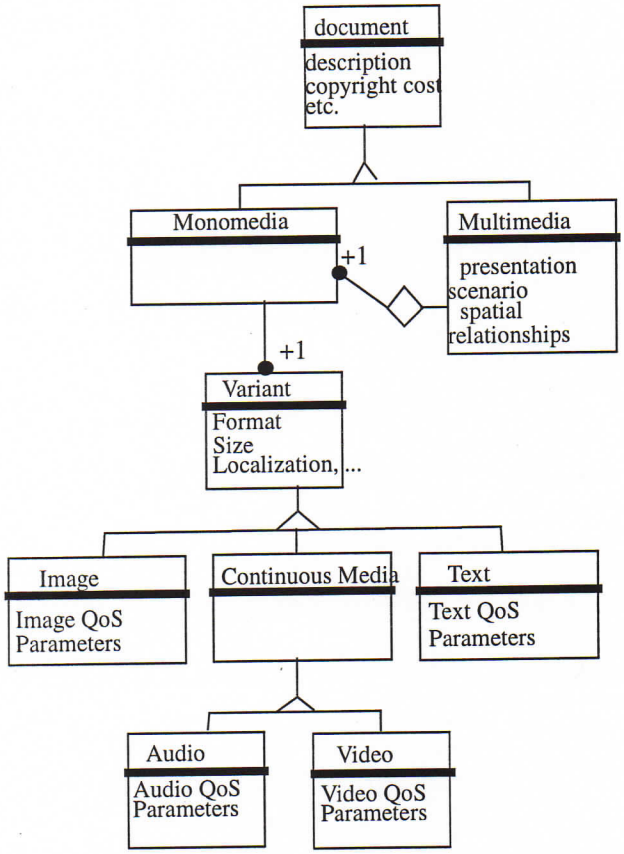

Figure 1. MM Document Model

A monomedia is defined in a particular medium: a text, a still image, an audio sequence, a graphic or a video sequence. Several physical representations, called variants, may exist for a monomedia object which correspond to a format variant. Each variant has different degrees of qualities. More generally, variants of the same monomedia may differ in terms of some static parameters which concern mainly the format of the coding, the size of the file, the QoS parameters associated with the file, e.g. video color and audio quality, and the localization of the file. For example, two variants of the same video sequence could offer different color qualities; Variant1 could be a super-color variant of the video sequence, while Variant 2 could be the black and white variant of the same video. Copies of the same file are considered also as variants. More detailed description of the model of MM document used can be found in [1, 5].

We define meta-data of a MM document as the data about the representation, the structure, QoS, the storage and the variants of the document. Meta-data is used to

locate, access, deliver and present the MM document. The document model shown in Figure 1 represents the meta-data associated with a MM document.

The QoS manager needs the meta-data of the MM document to play in order to perform QoS negotiation. Indeed, upon receipt of the user request to play a MM document with the desired QoS, the QoS manager determines, for each monomedia of the document, which of the available variants is the "best". This means that the machines that will deliver the variants are determined. Obviously, this activity requires the knowledge of the meta-data (location, QoS characteristics...) associated with the document to play. A detailed description of the negotiation procedure can be found in [5].

3. Meta-data and WWW

We assume the existence of a QoS manager that is responsible for QoS management functions. The operation of the QoS manager can be initiated either by a helper application, plug-in, or a Java applet.

Helper Applications

Helper Applications or External Viewers are the external programs used by the Web browsers to display the different types of media document [6]. These tools interpret data like audio and video that a normal browser does not support. Each Helper Application has an associated MIME type and particular assigned file extension. Each document containing a media of that MIME type should have the corresponding extension. Web browser, like Netscape, proceed by activating the appropriate Helper depending on the file extension of the document matched during the browsing.

Plug-ins

A Plug-in is a specific code for a particular platform that is designed to extend Netscape Navigator to include a wide range of interactive and multimedia capabilities [12]. Plug-ins are associated with one or more MIME types used by the browser to activate them. When activated, the browser enumerates the available plug-ins on the platform, and registers each plug-in with the corresponding MIME type. An instance of a Plug-in is activated each time a corresponding data MIME type is encountered. This instance is killed when the user quits the corresponding page or when he/she closes the holding window. Plug-ins communicate with Netscape via an API (Application Programming Interface).

Java applet

A Java applet is a specific kind of application in Java bytecode which can be transmitted over the network and executed within WWW Browser. This means that applets are applications down-loadable on demand. An applet can use only its own Java code and the Java API the applet viewer provides. More restrictive, an applet can't read or write files on the host that's executing it.

The main constraint associated with Plug-ins and Helper Applications is that they should reside on the client machine. Furthermore, they are (generally) hardware platform/operating system/window system dependent which is not the case for Java applets. However, Plug-ins and helper applications do not have the security restrictions of applets.

Upon the receipt of a user request (e.g., clicking on a button that represents a video/audio document), the QoS manager operation is initiated by one the technologies described above. The main issue we deal with is how the QoS manager gets the meta-data associated with the MM document (essentially video/audio) in order to perform the management operations, particularly QoS negotiation [5]. We identify two solutions: (1) Store meta-data in a database (or system files, e.g. Unix files), we call meta-data server; or (2) define some extensions of HTML to integrate meta-data directly in HTML documents.

3.1. Meta-data in a database or a file

To get meta-data, the QoS manager should communicate (e.g. via RPC) with the meta-data server (Figure 2). The meta-data server may consists of as much as complex as an object-oriented database or as simple as Unix files. The meta-data server can be located anywhere in the system, particularly on the Web server or on the host machine; it can be centralized or distributed. To support this architecture, the meta-data server should provide a standard interface to the QoS manager; this means that the nature of the meta-data server should be transparent to the QoS manager. Then, if we change the implementation of the meta-data server, we do not have to modify the code of the QoS manager.

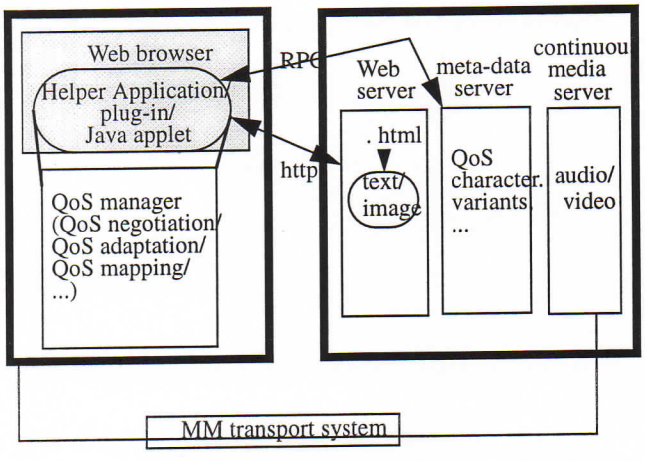

Figure 2. Meta-data in a meta-data server

In [4], we defined a set of primitives that should be supported by a meta-data server; we did implement these

primitives in the context of our CITR project [17, 4], where the meta-data server was realized using Objectstore.

(1) **GetAllMonomedia** (*Document, List of Monomedia*): *Status*
This primitive returns *List of monomedia* components of *Document*. It returns *Status* to indicate the success or the failure of the operation.

(2) **GetQoSMonomedia** (*Monomedia, QoS parameters*): *Status*
This primitive returns the QoS parameters of *Monomedia*. Examples of these parameters are the type of monomedia, e.g., video or audio, and the price associated, e.g., copyright.

(3) **GetAllVariants** (*Monomedia, List of variants*): *Status*
This primitive returns *List of variants* associated with *Monomedia*. It returns *Status* to indicate the success or the failure of the operation.

(4) **GetQoSVariant** (*Variant, QoS parameters*): *Status*
This primitive returns the QoS parameters associated with *Variant*. It returns Status to indicate the success or the failure of the operation.

(5) **GetSize** (*Variant, Size*): *Status*
This primitive returns *Size* of *Variant*. It returns *Status* to indicate the success or the failure of the operation.

(6) **GetFormat** (*Variant, Format*): *Status*
This primitive returns *Format* of *Variant*. It returns *Status* to indicate the success or the failure of the operation.

(7) **GetSite** (*Variant, Location*): *Status*
This primitive returns *Location* (the address of the file server which stores *Variant*) of *Variant*. It returns *Status* to indicate the success or the failure of the operation.

(8) **GetPresentationScenario** (*Document, Presentation scenario*): *Status*
This primitive returns *Presentation scenario* associated with *Document*. It returns *Status* to indicate the success or the failure of the operation.

3.2. Extension of HTML to integrate meta-data

As stated above, meta-data are necessary for the QoS management activity. As we are in the Web environment and the text and image parts of the multimedia documents are already coded in HTML, we propose to directly include the meta-data in the HTML documents (Figure 3). For the needs of multimedia and QoS meta-data we have investigated various proposed HTML extensions. We have retained two newly defined tags (<RESOURCE> and <OBJECT>) that seem suitable for the description of the meta-data and the definition of multimedia (audio and /or video) document structures in HTML [10]. The description of the <RESOURCE> and <OBJECT> elements can be found in [8] and [7].

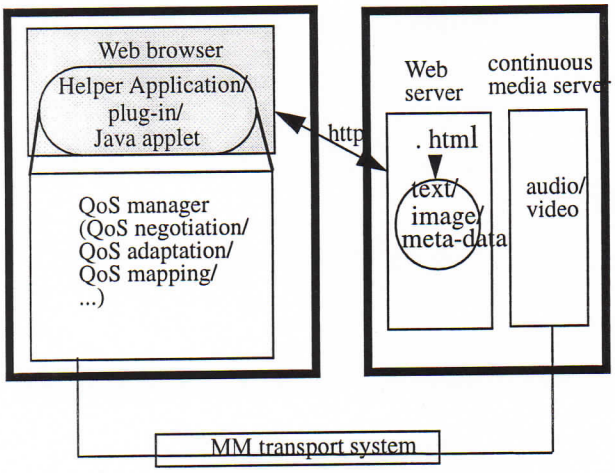

Figure 3. Meta-data in (extended) HTML document

The <OBJECT> tag allows the HTML author to specify the data and /or the properties/parameters for initializing objects to be inserted into HTML documents as well as the code that can be used to display/manipulate that data.This definition is well suitable for our need to include multimedia object (video and/or audio) with facilities to specify the parameters (meta-data) and indicate the code that handles (QoS negotiation and media displaying program) the multimedia object. We use the <RESOURCE> element to deal with the description of resource variants. It provides information on some resource within another HTML document. The <LINK> elements, when used with their REL attribute enable the description of the resource variants. The resource is then a generic resource while variants are specific resources. Each <LINK> element describes a resource variant. The relationships of the <LINK> element provide a mean to indicate how resource variants differ from one another i.e the link type provides a mean to specify what parameter of quality varies.

We have used <RESOURCE> to code the meta-data [10]. Since the above described meta-data are application specific, we have used their experimental format, which means that parameter names are preceded by "x-". For the description of the resource variants, none of the existing relationships seems to fit our needs. In fact, the variants may differ by a combination of QoS parameters instead of a single QoS parameter. So, we propose a "QoS-specific" relationship to simply indicate that variants have several QoS characteristics. Given these semantics, we can now describe meta-information on our multimedia objects and insert them into a HTML document. Suppose the multimedia document is composed of a text part coded in HTML, an video having only one variant and an audio available in two variants. We propose the following description of the meta-data for the audio and the video in HTML:
<resource href = Audio_monomedia>

<link rel = QoS-specific href = audio1>
<link rel = QoS-specific href = audio2>
</resource>
<resource href = audio1>
<meta http-equiv = url value =" pnm://audio.realaudio.com/audiofile1.ra">
<meta http-equiv = x-format value = g728>
<meta http-equiv = x-duration value = 16>
<meta http-equiv = x-quality value = phone>
</resource>
<resource href = audio2>
<meta http-equiv = url value = " pnm://audio.realaudio.com/audiofile2.ra">
<meta http-equiv = x-format value = g728>
<meta http-equiv = x- duration value = 20>
<meta http-equiv = x-quality value = cd>
</resource>
<resource href = video>
<meta http-equiv = url value = "cmfs://sitename/UIO-texte-file">
<meta http-equiv = x-format value = h261>
<meta http-equiv = x-duration value = 10>
<meta http-equiv = x-framerate value =30 >
</resource>

To insert the multimedia object in HTML,

<OBJECT id = "multimedia-object" classid = "negotiation-prg">
<PARAM name = audioComponent value = "Audio-monomedia">
<PARAM name = videoComponent value = "video">
</OBJECT>.

A more detailed description of these extensions can be found in [10].

4. Implementation

The Web browser we used for the implementation is Netscape.

4.1. Using helper application technique

The technique of the Helper Application was used as a first solution to produce a Web-compatible version of our News-on-Demand prototype [17, 4]; It consists of simply using the initial CITR prototype as a Helper Application. We coded text and image components of the multimedia documents in HTML and stored meta-data as a simple ASCII file; let us note that in the original prototype, the QoS module gets the meta-data from the distributed multimedia database (DBMS) which is based on Objectstore. Video and audio are included in HTML as hypertext links which point to the related meta-data (Figure 4). An experimental MIME type with the corresponding "meta" extension is associated to the meta-data. The user, when clicking on such a link, activates the QoS negotiation module of the CITR prototype. The program gets the meta-data file name as argument instead of getting them from the multimedia database. From this moment, the web version of the News-on-Demand system behaves like the original version.

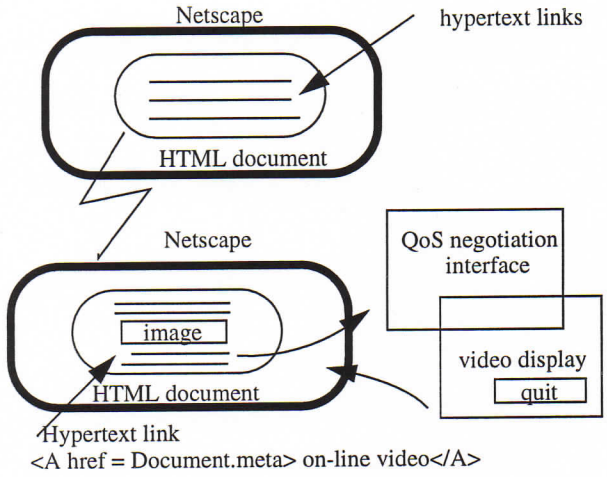

Figure 4. An implementation of the extended version of HTML

The client software is globally composed of a Web browser (Netscape), the QoS module, a synchronization module which accesses the CMFS server; the server part is made of the HTTP server and the CMFS server (Figure 5).

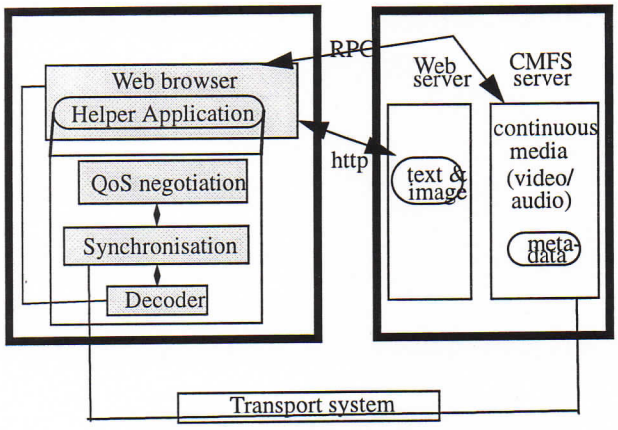

Figure 5. Architecture of the news-on-demand system implemented as helper application

4.2. Using Java applets

Based on our experience with the CITR prototype, we implemented a user agent which understands the HTML-encoded meta-data described in Section 3; we decided to go for a Java solution. Before the access, transfer and display of the video, some initialization activities must take place:

1- To get the meta-data

2- To get the user QoS preferences via an user interface or the use of an existing user profile.
3- To perform QoS negotiation in order to choose the best variant for the user that can be supported by the system.
4- If such a variant exists, display it.

As <OBJECT> and <RESOURCE> are not yet supported, we activate the object via a Java applet that receives as parameter, the identifier (id) of the multimedia object. The applet then parses the HTML code in order to find the object and associated meta-data in the extended HTML page and performs the necessary processing. The meta-data must be in the same HTML document with the applet. The important classes used for this activity are:

- MM_Document, a class that represents a structured multimedia document as shown in Figure 1;
- PageParser, a Java-based HTML parser.

The applet's parameters are as follows:

- <APPLET CODE="ExtendApplet.class" CODEBASE = "http://www.iro.umontreal.ca/~madja/sometest" WIDTH= 1000 HEIGHT= 600>
<PARAM name = "METADATA_ID" value = "multimedia-id">
</APPLET>

The applet then creates an instance of the class MM_Document with the object Id and the URL of the HTML document (containing the meta-data) as parameters. MM_Document class makes use of PageParser class to build the multimedia document structure.

The user may specify his/her desired values for the QoS parameters via a graphical user interface. This graphical interface is the visible part of the applet. For video, we have retained three QoS parameters at the user level. They are: frame rate, resolution and color. For audio, we have audio quality (Phone or CD) and language. The cost is also provided as a choice criteria. The user may also define his/her priority among the QoS parameters, on one hand, and on the two monomedia, on the other hand. As an example of QoS specification by the user, we consider the following:

FrameRate >=10
Resolution >= medium
Cost <= 1$
Priority1 = Cost
Priority2 = FrameRate
Priority3 = Resolution

The priority system enables classification among variants. Giving a user profile and the available variants, the QoS negotiation engine chooses the best variant that the system can support at the given time; in case of a problem, an adaptation will be made by considering the next best variant. For more details on these operations see [5]. Figure 6 shows the main user interface; the user's preferences are expressed in terms of a profile that models all the information we need to make a choice for the user. The user can use a predefined default profile or set a profile that can be composed of more than one sub-profiles. Each sub-profile is a set of user's QoS parameters values and Cost. Cost is the cost the user is willing to pay for the QoS level defined by these values. Precisely, the specification described above constitutes one sub-profile. For each sub-profile we create an instance of the class UserProfileClass. The user's profile is stored in a file for later use.

The Negotiation activity is handled by the Negotiation class. The Negotiation class uses the document structure and the user's profile as parameters. The negotiation consists of:

1- A static negotiation that makes a comparison of media variants against the client machine characteristics. These characteristics are modelled by the class ClientMachineClass. All media variants that can't be supported by the client machine are discarded.
2- A comparison of the remaining variants against user's profile.
3- A sorting of the variants according to their suitability to the user's profile and his priority among QoS parameters
4- The QoS manager then requests the first element of the sorted list. If the requested variant cannot be supported by the server, the following offer is considered and so on. The URL of the variant indicates the server from which the variant should be requested.

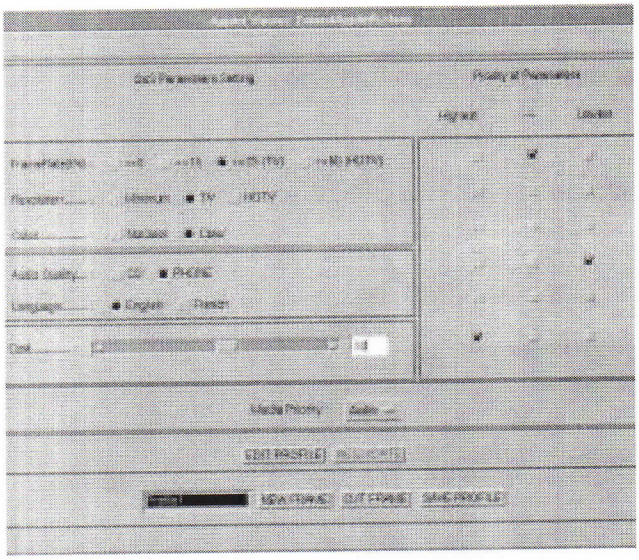

Figure 6. The main user interface of the web-compatible news-on-demand prototype

The original version of our the News-on-Demand system uses the CMFS server for accessing and retrieving continuous media. Our Web version provides this facility. However, we also intend to include other continuous media servers existing in the Web environment: The Vosaic server [16], the RealAudio/RealVideo servers[14]. Vosaic is a cli-

ent-server system developed at the University of Illinois that provides real-time video on the Web. It uses the VDP (Video Datagram Protocol) protocol for continuous media transfer. VDP uses an adaptation algorithm to find the optimal transfer bandwidth. RealAudio developed by Progressive Networks, provides high quality audio over the Web. Progressive Network announces recently their RealVideo System.

It would be interesting to integrate these various servers such that different media variants may use different communication protocols for real-time presentation, corresponding to the available technologies (Figure 7). The non-continuous parts of the multimedia document could be provided by the existing HTML servers. The client software will consequently be an integration of the client software for each of these different servers.

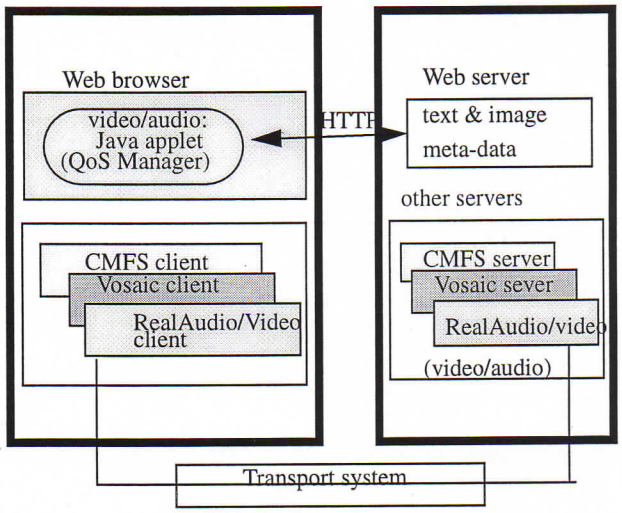

Figure 7. Architecture of the news-on-demand system for the Web

5. Conclusion

The aim of our work was to integrate meta-data of multimedia documents in WWW; this was motivated by the fact that QoS management needs meta-data of the corresponding MM documents. The meta-data we used have been defined in an ongoing CITR major project "Broadband Services" [17]. A first solution (we adopted) was to store meta-data on a database or file systems; we were not happy with this solution, since MM documents (HTML document) were stored and accessed "independently" of their meta-data. This pushes us to find the HTML extension conveying the meta-data related to QoS issues. Fortunately, we didn't have to define new HTML tags. <RESOURCE> and <OBJECT> seem suitable for our need. Before they can be supported by the popular browsers, we have implemented a Java applet which handles it. There are a number of other working groups that investigate real-time audio and video for the Web with QoS management support, such as the FastWeb project [3]; but a QoS negotiation that involves the user was not addressed apart from the FastWeb project.

References

[1] G.v.Bochmann, B.Kerherve, A.Hafid, P.Dini and A.Pons, Functional Design of Adaptive Systems, In Proceedings of the IEEE Workshop on Multimedia Software Development, Berlin, Germany 1996

[2] E.Dubois, N.Baaziz and M.Matta, Impact of scan conversion methods on the performance of scalable video coding, In Proceedings of IS&T/SPIE, San Jose, February, 95

[3] M.Fry, A.Seneviratne, A.Vogel and V.Witana "Delivering QoS Controlled Continuous Media on the World Wide Web", In Proc of IWQoS96, Paris, March 1996

[4] A.Hafid and G.v.Bochmann, QoS Management in News-on-Demand Systems: Implementation, In the Proceedings of the Third International Workshop on Protocols for Multimedia Systems, Madrid, Spain, 1996

[5]A.Hafid and G.v.Bochmann, A Quality of Service Management Approach: Design and an Implementation, Multimedia Tools and Applications Journal, 1998 (to appear) (http://www.csd.uwo.ca/faculty/hakim/Publications.html)

[6] Helper Applications, http://www.netscape.com/assist/helper_apps/

[7] Giving Information About Other Resources in HTML, http://www.w3.org/pub/WWW/Mark-up/Resource/Specification

[8] Inserting objects into HTML, http://www.w3.org/pub/WWW/TR/WD-object.html

[9] L.Lamont and N.D.Georganas, Synchronization Architecture and Protocols for a Multimedia News Service Application, In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Boston, May 1994

[10] E.Madja, G.v.Bochmann, R.Dssouli and J.Gecsei, HTML extensions for Multimedia Documents and Quality of Service management on the Web, Submitted for Standardization

[11] G.Neufeld, D.Makaroff and N.Hutchison, Design of a Variable Bit Rate Continuous Media File Server for an ATM Network, In Proceedings of IS&T/SPIE'96, San Jose, California,1996

[12] Plug-in guide, http://home.netscape.com/eng/mozilla/3.0/handbook/plugins/pguide.htm

[13] J. Rambaugh & al. Object Oriented modelling and Design, Prentice Hall, 1991

[14] RealAudio, http://www.realaudio.com/

[15] C.Vittal, M.Ozsu, D.Szafron, G.Medani, The Logical Design of a Multimedia Database for a News-On-Demand Application, Technical Report #94-16, University of Alberta, Canada, 1994

[16] Vosaic, Continuous Media on the Web, http://choices.cs.uiuc.edu/Vosaic/Vosaic.html

[17] J.Wong, K.Lyons, R.Velthuys, G.Bochmann, E.Dubois, N.Georganas, G.Neufeld, T.Ozsu, J.Brinskelle, D.Evans, A.Hafid, N.Hutchinson, P.Inglinski, B.Kerherve, L.Lamont, D.Makaroff, and D.Szafron, Enabling Technology for Distributed Multimedia Applications, IBM Systems Journal, Volume 36, No 4, 489-507, 1997