Chapter 4

THE REMOTE MULTIMEDIA DOCUMENT RETRIEVAL SYSTEM (ReMDoR)

4.1 Introduction

This chapter describes ReMDoR, a remote multimedia document retrieval system that allows authors to specify synchronization requirements and varying degrees of reliability for multimedia elements (Conrad et al., 1996; Conrad et al., 1998; Caro, 1998). As explained in Chapter 1, the motivation for ReMDoR was the investigation of partially-ordered/partially-reliable transport protocols in the context of multimedia document retrieval. Because it was infeasible to incorporate partial order and partial reliability into an existing multimedia document retrieval system, the author developed ReMDoR, a prototype multimedia document retrieval system to carry out this research.

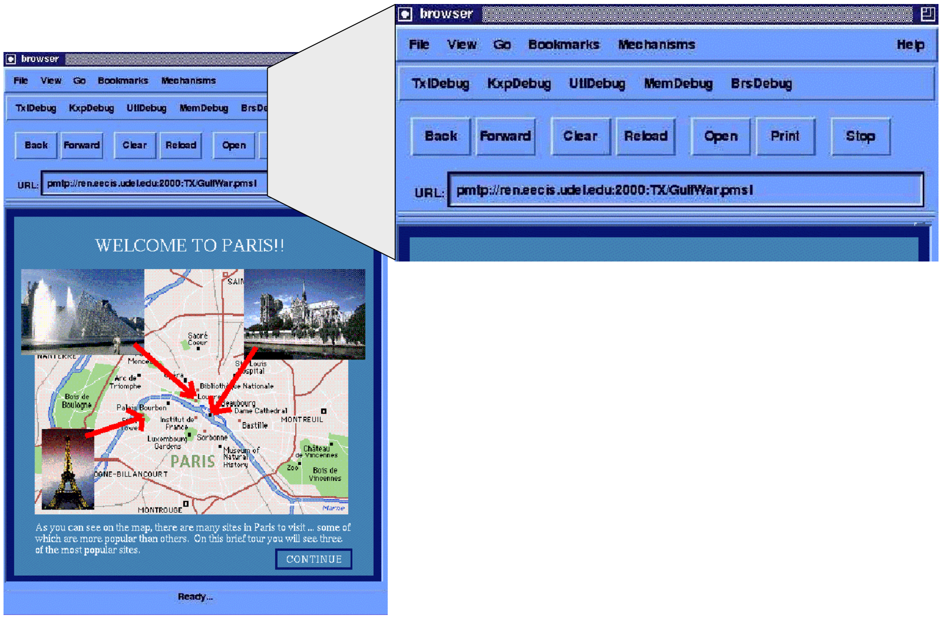
- Section 4.2 overviews ReMDoR's functionality.
- Section 4.3 describes the main components of ReMDoR's architecture by comparing and contrasting this architecture with that of the World Wide Web.
- Section 4.4 describes the syntax and semantics of the language used for specifying documents in ReMDoR.
- Section 4.5 describes the client/server protocol used in ReMDoR.
- Section 4.6 describes the ReMDoR compiler.
- Section 4.7 surveys related work on multimedia systems.

- Section 4.8 describes the history of the ReMDoR project.
- Section 4.9 provides a chapter summary, and suggestions for future work.

4.2 Overview of ReMDoR functionality

ReMDoR's basic model is similar to that of the World Wide Web; documents are available on a server and are retrieved via a browser. Figure 4.1 shows the user interface of the ReMDoR browser, which is similar to that of familiar web browsers. However, unlike Web documents, ReMDoR documents are *temporal*—they have a time dimension requiring synchronization of elements such as audio, video, still-images, text, pauses, and interactions.

ReMDoR has capabilities that support experimentation with innovative protocols and data compression techniques, such as (1) the ability to select from a wide range of transport services and transport service features (via UTL), (2) the ability to record statistics about performance on an object-by-object basis, (3) features to automate repeated performance experiments, and (4) the ability to easily incorporate new image formats, such as formats required for network-conscious image compression research (Iren, 1999).



pmtp://ren.eecis.udel.edu:2000:TX/GulfWar.pmsl

As you can see on the map, there are many sites in Paris to visit ... some of which are more popular than others. On this brief tour you will see three of the most popular sites.

Figure 4.1 User interface of ReMDoR browser

4.3 Overview of ReMDoR system components

In this section we explain the architecture of ReMDoR by comparing and contrasting it with the architecture of the World Wide Web. Figures 4.2 and 4.3 show the architectures of these two systems.

Figure 4.2 highlights the four basic components of the Web architecture:

(1) the HTML ***language*** for specifying the structure of documents,

(2) the Web ***server***,

(3) the HTTP ***protocol*** for transferring documents, and

(4) the web ***browser***.

By contrast, Figure 4.3 illustrates that the ReMDoR architecture has six components: (1) a language, (2) a document compiler, (3) a file format (4) a server, (5) a protocol, and (6) a browser. Here are those six components again, in more detail:

(1) The syntax and semantics of **a *language*** for specifying multimedia documents. We call the language used to specify ReMDoR documents ***Prototype Multimedia Specification Language* (*PMSL*)**. PMSL was introduced in (Conrad et al., 1996).

(2, 3) A ***document compiler*** that takes a PMSL document as input, parses it, and produces as output a compiled multimedia document in the ***Prototype Multimedia File Format (PMFF).*** A PMFF file is an ASCII file containing information that the server can use to efficiently packetize the multimedia information, and send it via a partially-ordered transport protocol. While a PMFF file is human-readable, it has a particular structure that would be difficult for a human to produce manually—thus, the need for the compiler.

The compiler and PMFF file format are the two extra components in ReMDoR's architecture that do not correspond to any component of the Web architecture. In the Web, all parsing of HTML occurs at the client only; the server does nothing more than read to a request for a particular file, read that file from disk, and send the contents (uninterpreted) over the wire. In ReMDoR, by contrast, because PO/PR transport service is used, the file retrieved by the server must contain a representation of the partial order. Because POCv2 requires the partial order to be in transitively reduced form, which takes $O(n^{(2.36)})$ to compute in theory, but $O(n^3)$ time in practice[44], we designed ReMDoR to perform this processing off-line rather than in real-time as the document is requested.

(4, 5) A ***server*** that can respond to requests for documents using a ***protocol***. We call our protocol ***Prototype Multimedia Transfer Protocol (PMTP***), by analogy with HTTP. Our server is similar in role to Web servers. However, unlike Web servers that send documents uninterpreted over the wire, the PMTP server reads a PMFF file, and converts the contents to PMTP. Fortunately, the transformation from

---

[44] See Chapter 6 for detailed analysis.

PMFF to PMTP is trivial; essentially, PMFF is nothing more than an efficient way to represent a service profile and the headers of PMTP packets in an ASCII file. The remainder of the PMTP packets, which consists of image or audio data, is read directly from the audio or image files by the server. Just as in HTML/HTTP, an image that appears in more than one document, or more than once in a document, need be stored only once at the server. However, unlike early versions of HTTP, the PMTP protocol does not require that multiple connections be established if multiple images appear in a document; all data is sent over a single connection. The ability to send multiple images over a single connection is a feature that has only recently been incorporated into HTTP 1.1.

(6) A ***browser*** providing a GUI allowing the user to request from a server a PMFF file from a server via its URL. The browser parses a URL specification of a PMFF document, formats it as a PMTP request, sends the request to the server, interprets the data returned as multimedia elements: text, audio, vector graphics and bitmap graphics (images), and presents elements to the user.

4.4 Syntax and Semantics of PMSL

PMSL provides a document author with a means to specify the elements that make up a document along with their spatial and temporal layout. Spatial layout for graphic elements is specified using an (x,y) coordinate system with (0,0) representing the upper right corner. Temporal layout is specified by means of a partial order, using the explicit release mechanism to emulate the semantics of OCPN (See Sections 2.5 and 2.6).

PMSL documents consist of definitions and elements. There are four kinds of definitions:

- color definitions (keyword: COLORDEF), which assign an identifier to a color (e.g., TitleColor).
- font definitions (keyword: FONTDEF), which assign an identifier to a font (e.g., labelFont)

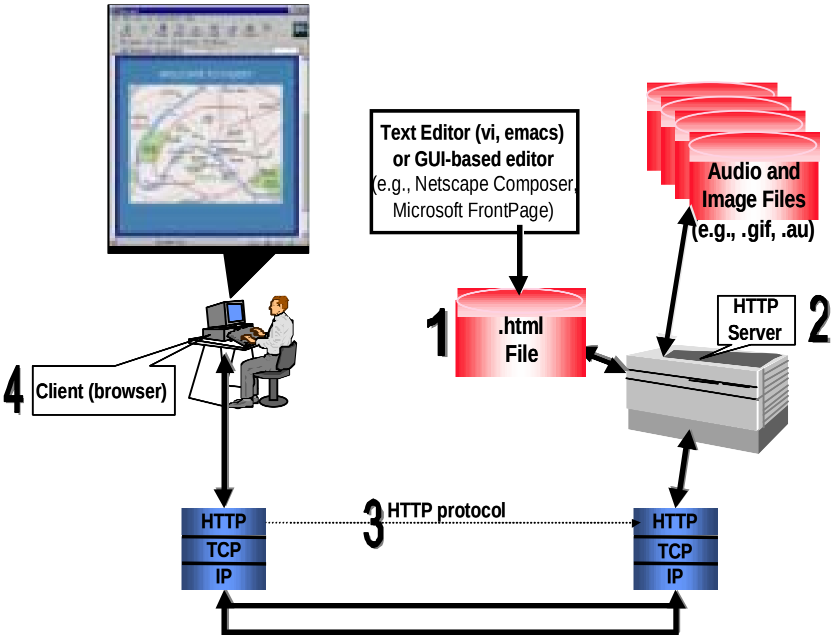


Figure 4.2 Web architecture, for comparison with ReMDoR.

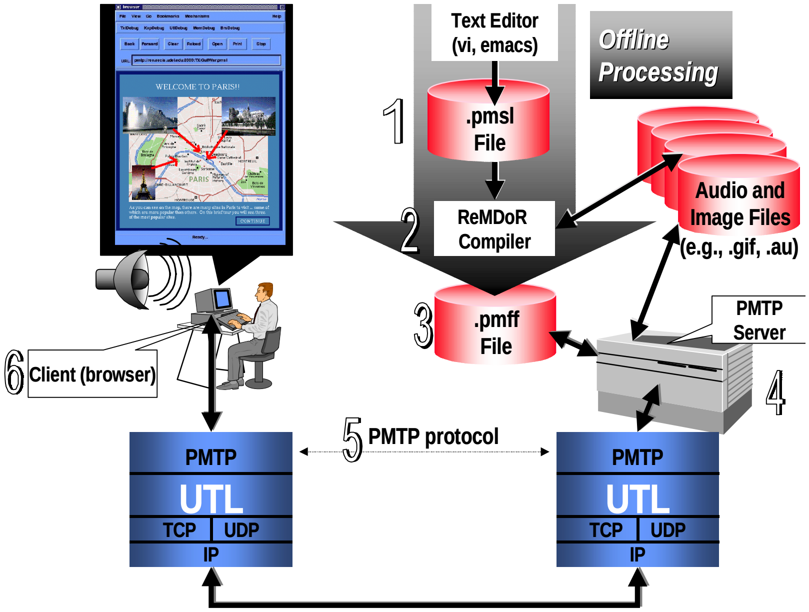


Figure 4.3 ReMDoR system architecture

- pen definitions (keyword: PENDEF), which assign an identifier to a set of drawing attributes (e.g., arrowPen). Pens in PMSL correspond precisely to the concept of a *graphics context* in the X-Windows system (Gettys et al., 1987**;** Nye, 1992)

- text formatting definitions (keyword: TXTFORMATDEF) which associate an identifier (e.g., captionFormat) with a particular text layout, including left margin, right margin, top margin, line spacing, and justification.

Currently, there are six kinds of elements, each introduced with the keyword ELEMENT.

- graphic elements which may be either:
    - text, using any color or font support by X-Windows,
    - bitmapped images(in one of four supported formats), or
    - vector graphics (e.g., boxes or lines.)
- audio elements, currently limited to audio clips in Sun `.au` format (8Khz µ-law encoding),
- pause elements, which introduce explicit pauses in the document,
- continue elements, which allow for user interaction,
- hotspots, which provide hyperlinks to other PMSL documents,
- erase elements, which remove graphic elements from the display, and
- null elements, which can be used to structure a partial order.

The appendix contains examples of PMSL documents, while (Caro, 1998) contains the Backus-Naur Form grammar for the PMSL language.

4.5 The PMTP protocol

PMTP uses an ASCII-based protocol, following the example of the Internet protocols such as HTTP, FTP, SMTP, and NNTP. While less efficient in terms of bandwidth, an ASCII protocol does provide several advantages. Our chief motivation was the improved ease of debugging. Also important was the fact that ASCII provides a common presentation layer format that does not depend on using the network-byte-order conversion routines[45], since correct handling of network-byte-order is a notoriously error prone process.

The appendix contains examples of PMTP messages, and the way in which these are stored in a PMFF file. Here, we provide an overview of the PMTP language.

There are five message types in PMTP. The first type is sent only from client to server:

(1) **request** messages consist only of the name of a PMFF file to be retrieved.

The remaining four message types are response messages sent only from server to client. The delivery of these message types is governed by partial order delivery. The first of the response message types is:

(2) **pen** messages, which correspond to PENDEFs in PMSL

The remaining three are all messages that correspond to ELEMENTS in PMSL:

---

[45] Since TCP/IP lacks a standard presentation layer, and since network byte order on the Internet is, by convention, big-endian, byte swapping macros are provided for encoding all binary fields that are placed in, or retrieved from PDUs transmitted over the Internet. Examples of these routines are `htonl()` and `ntohl()` (host to network long, and network to host long, respectively). These routines perform the necessary swap on little endian architectures, and reduce to no-ops on big-endian architectures.

(3) **blok** (block) messages correspond to PMSL elements that can be sent in a single packet. These include the following PMSL element types: PAUSE, NULL, END, TEXT, LINE, BOX, ERASE.

(4) **cell** messages correspond to PMSL elements where the element data does not fit into a single packet. When the UTL mechanism selected supports it, cell messages are sent using the stream abstraction described in Section 2.3. Cell message are used for IMAGE and AUDIO elements.

(5) **cend** (cell end) messages are a special case of cell messages; the cend message indicates that the message is the last cell in a particular stream; apart from indicating the end of the stream, cend messages have the same syntax and semantics as cell messages.

Each of the server-to-client messages contains a PMTP header formatted in ASCII, and for some PMTP response messages (specifically, IMAGE, AUDIO and TEXT messages), a trailer containing additional data. In the case of IMAGE elements, the trailer contains data formatted according to the particular image format (GIF, NCG, NCW or WVT). For AUDIO elements, the trailer contains audio samples in 8Khz μ-law format. For TEXT elements, the trailer contains the text to be displayed.

4.6 Functions provided by the ReMDoR document compiler

The input to the ReMDoR document compiler is an ASCII file in the PMSL language described in Section 4.4. In addition there are four optional parameters related to the allocation of bandwidth to audio elements; we reserve a discussion of these parameters to the description of the linear extension selection algorithm in Chapter 6. The output is a PMFF file that can be used by the server to respond to PMTP requests. The ReMDoR document compiler performs the following actions to produce this file.

(1) It parses the PMSL and extracts the partial order.

(2) It verifies that the partial order specified by the document author is consistent (i.e., the corresponding directed graph is acyclic) and it computes the transitive reduction.

(3) It then creates a directed graph structure corresponding to the partial order, and selects a linear extension. The linear extension selection algorithm is based on the topological sorting algorithm suggested by Exercise 23.4-5 in (Cormen et al., 1990), modified to take into account the POCv2 stream abstraction, and the need to guarantee audio elements with a minimum bandwidth of 64kbps. This algorithm is described in more detail in Chapter 6.

(4) As the linear extension is computed, a PMFF file is created. The PMFF file begins with a representation of the partial order using the format defined in Section 2.4.3. It then contains the headers of each of the PMTP packets, preceded by fields containing the information the server needs to properly send the data using UTL, such as the object number, and the total packet length.

For objects with a trailer, information concerning the trailer follows each header in the PMFF file. There are two possibilities. For TEXT elements, the contents of the trailer appear directly in the PMFF file. For image and audio elements, the PMFF file contains a filename, offset and length of the trailer data; the server uses these fields to read the data from the image or audio file directly into the outgoing packet.

(5) Finally, an optional flag on the ReMDoR compiler can be used to produce a graphical representation of the partial order in PostScript format. It is a difficult problem to lay out a partial order graphically in a conflict-free manner; that is, in a way that that lines representing edges do not inadvertently intersect vertices on which they are not incident. Rather than trying to use a sophisticated approach, we use a simple heuristic[46] along

---

[46] Our heuristic has four steps: (1) sort the PO by level (i.e. partition the PO into antichains $A_1, A_2, \ldots A_k$ s.t. each element *x* of antichain A_i has the same depth in the PO, where depth is defined as the length of the longest chain of predecessors of *x*.) (2)

with multiple colors and multiple dash patterns to assist the user in avoiding any incorrect interpretation when edges touch vertices that they should not touch.

4.7 Related work[47]

There are many examples of systems for creating multimedia documents (authoring systems) and client/server systems for making multimedia documents available over a network (multimedia document retrieval systems.) In this section we provide just two examples of these systems. It should be emphasized that the current role of the ReMDoR system is primarily a means to an specific end: to evaluate PO/PR transport service. Therefore we acknowledge that ReMDoR has limited capabilities when compared with the research systems described below, or with commercial multimedia authoring systems such as Macromedia Director or Macromedia Authorware[48].

---

construct equal size columns, where each column contains the elements of exactly one antichain (3) place the elements of each antichain A_i in column *i*. Vertically, place each element of antichain A_i equidistant from one another w.r.t the elements in that column (i.e., use the entire vertical space) (4) apply a fixed vertical offset to the even numbered columns (this seems to help avoid some collisions of edges with vertices upon which they should not be incident.). We make no particular claims about the goodness of this heuristic. An investigation of the considerable body of research on graph-drawing would make an interesting side-investigation, but was judged not to be central or even necessary to our work, since our drawing tool was merely an aid to debugging.

[47] Related work on multimedia synchronization was surveyed in Section 2.4

[48] www.macromedia.com.

4.7.1 MEDIADOC/MEDIABASE (Rody and Karamouch, 1995)

The notion of adding transport QoS to document specification schemes has been previously addressed in the MEDIADOC/MEDIABASE project (Rody and Karmouch, 1995) which has several features in common with our work. In both projects (1) a client/server system is used to serve multimedia documents over a network, (2) QoS (e.g., reliability) can be defined on a per object level, and (3) the transport layer assists with synchronization. Our work differs in both focus and architecture. The MEDIABASE/MEDIADOC project focuses on the design of "an advanced high-performance distributed multimedia information and communications system with a particular focus on document architectures, database models, communication and synchronization, and … storage." As such their architecture is quite sophisticated.

By contrast, our emphasis is on exploring the usefulness of PO/PR transport protocols; we use multimedia document retrieval mainly as an example application to explore this concept. Hence we limit the scope of our document model and application architecture to the minimal framework necessary to test certain ideas. Our hope is that by developing PO/PR mechanisms and demonstrating their utility for multimedia document retrieval, we can provide useful ideas to the developers of systems such as MEDIABASE/MEDIADOC.

4.7.2 Fiets (Rutledge et al., 1998), and HyTime, DSSSL and SMIL

(Rutledge et al., 1998) present several issues related to the mapping between the semantic relationships of objects in a multimedia presentation, and their eventual spatial and temporal layout. They argue that rather than nailing down the exact spatial and temporal layout of a document, it is better to represent the semantic relationships between objects, and allow the computer system to determine an appropriate spatial/temporal layout. They present their findings in the context of a system called Fiets (Foundation for Interactive Electronic Touring Systems.) Fiets is an example hypermedia application providing geographic and historic information about tourist attractions in the city of Amsterdam. Of particular interest is the fact that Rutledge et al. discuss the use of three standard SGML-based[49] standard multimedia specification languages in the construction of the Fiets system:

HyTime	Hypermedia/Time-based Structuring Language (ISO/IEC 10744 (1992); DeRose and Durand, 1994).
DSSSL:	Document Style Semantics and Specification Language (ISO/IEC 10179 (1994); DeRose and Durand, 1994).
SMIL	Synchronized Multimedia Markup Language (pronounced "smile") (W3C 1998).

HyTime is a language for specifying Hypermedia documents, including their spatial and temporal layout. DSSSL is a Scheme-like language for specifying transformations from one SGML-based representation of a document to another. SMIL is a more recent format for specifying coarse-grained synchronization for temporal media in Web documents; its recent incorporation into a commercial system from RealNetworks gives it a chance at more widespread acceptance than previous standards in this area have enjoyed.

---

[49] SGML is the Standardized General Markup Language. SGML is the parent standard for many standard markup languages, including the Hypertext Markup Language (HTML) used in the Web. The most familiar feature of SGML derived languages is the now familiar tag construct: for example, in any SGML derived language, `<FOO>how now brown cow</FOO>,` indicates that the text "how now brown cow" should be marked with the tag "FOO".

We considered using HyTime as the basis for the ReMDoR system, however we rejected it because of the extreme complexity of the HyTime syntax and semantics. SMIL, on the other hand, seems like a good candidate for forward progress with ReMDoR (Urbano et al., 1999.)

4.8 Project history

There have been two major versions of the ReMDoR system. The first version was developed by the dissertation author, with programming support from M.S. student Edward Golden, between summer of 1995 and spring of 1996. Mr. Golden's contribution was mainly the implementation of the parser for the PMSL syntax, and the original PMTP binary protocol (explained below.) The dissertation author provided overall project supervision, and also designed and implemented the browser, including the graphics routines, audio routines, routines for processing the partial order, and the user interface.

The second version was developed in conjunction with Armando Caro, who made it the subject of his undergraduate honors thesis, developed between spring 1996 and spring 1998 (Caro, 1998). In version two, two key changes in the architecture were made.

First, the initial version had only two components: a server, and a browser. The functionality of interpreting the syntax of PMSL and extracting the partial order was done entirely within the server. Because processing larger documents took several seconds, the author decided that in the new version, the processing should be done off-line. Second, the initial version of PMTP was a binary protocol. In PMTP v1,the information regarding the multimedia elements was encoded, for example, using an 8-bit field for the element type, 16-bit fields for x and y coordinates measured in pixels, etc. Version 2 of PMTP replaces this with an ASCII protocol for reasons already covered in Section 4.5. In addition to these changes, many ease of use features were added, and abstractions were added to the image processing routines to facilitate the addition of new image formats. Recently, Iren and Caro used these abstractions to add support for three additional images formats required for the NETCICATS project:

(1) Network Conscious GIF (Amer et al., 1999),

(2) the SPIHT Wavelet format
(Said and Pearlman, 1996; Iren, 1999), and

(3) the Network Conscious Wavelet format (Iren et al., 1998).

4.9 Chapter summary and suggestions for future work

In this chapter, we presented the ReMDoR system, which provides the basis of our performance evaluation of PO/R transport service. We described the major components of the system, including the language used to author documents, the ReMDoR protocols and the key functions of the ReMDoR compiler.

Future work on ReMDoR includes porting it to the Linux operating system, adding capabilities for video, and implementing a version of ReMDoR in the Java language to allow for greater portability.

In the next chapter, we describe the results of performance experiments based on UTL and ReMDoR.