Chapter 1

INTRODUCTION AND MOTIVATION

1.1 Transport Layer

The transport layer is the lowest of the layers in the ISO (International Standards Organization) reference model responsible for end-to-end quality of service (QoS) in a packet-switched network. Re-sequencing out-of-order data, recovery from data loss, detecting and removing duplicates, and flow control/congestion avoidance are among the typical functions of the transport layer. A QoS parameter of the transport layer is what determines the level of service provided by each of these functions.

Conrad et al. [1] explains that sometimes authors lump three QoS parameters (loss, order, and duplication) together under the term "reliability", using "reliable" to refer to a transport service where no messages are lost, delivered in a re-arranged order, or duplicated. Transport QoS can be classified with greater precision if "reliability" strictly refers to data loss, defining a reliable service as one that allows no loss whatsoever. This more precise definition of reliability makes it independent from the QoS parameters: "order" and "duplication". "Order" refers to the extent to which the transmitted sequence of elements is preserved in delivery. An ordered service delivers elements in the exact same order as transmitted, while an unordered service makes no such guarantee about order. "Duplication" refers to whether or not multiple copies of the same element may be delivered. A no-duplicates service

detects and discards any duplicates, while a maybe-duplicates service does not make such a guarantee [2].

Flow control is the final QoS parameter of the transport layer. This parameter refers to the effort put into avoiding network congestion by controlling the flow of data during a connection. A flow-controlled service has a mechanism for smoothing out the burstiness of data transmission, while a service that is not flowed-controlled has no such mechanism [3].

1.2 Traditional Protocols

Today's Internet provides a choice between two transport protocols: Unit Datagram Protocol (UDP) [4] and Transmission Control Protocol (TCP) [5]. These protocols present extremes in terms of QoS. UDP's service is unordered, unreliable, maybe-duplicates, and is not flow-controlled; TCP's service is ordered, reliable, no-duplicates, and is flow-controlled.

1.3 Internet Application Developer's Dilemma

The QoS needed by different applications varies greatly, and the fact that traditional protocols only provide the extremes creates a dilemma. Today's developers typically have three choices when designing an application for the Internet: (1) use UDP, (2) use TCP, or (3) use UDP and build the additional needed transport functionality as part of the application development effort. When an application has to choose either UDP or TCP when neither is appropriate, negative consequences result. If TCP is chosen for an application that does not require total order and/or full reliability, unnecessary delays in information delivery may result. If UDP is used to transmit vital information, important data may be lost or misordered unless the

application incorporates the complexity to provide order and reliability. However, designing and implementing an application specific transport protocol that correctly handles retransmission and flow control may be a larger effort than designing and implementing the application itself. Hence, applications need a flexible transport service that allows applications to receive the QoS they require [6].

1.4 POCv2 as a Solution

A flexible transport service offering a partially-ordered, partially-reliable (PO/PR) service is ideal for applications that need flexible control over the ordering and reliability of individual elements [6]. Such a service is essential for balancing various QoS parameters in different applications, while allowing to build upon previous work rather than duplicating efforts. This approach is consistent with Application Level Framing as proposed by Clark and Tennenhouse [7].

To address this need, the Protocol Engineering Lab (PEL) at University of Delaware has developed and implemented a new transport protocol, Partial Order Connection version 2 (POCv2), which provides a spectrum of PO/PR services. Figure 1 illustrates the entire spectrum of PO/PR services available and shows where UDP and TCP exist in the spectrum. POCv2 is a transport service ideal for applications, such as multimedia applications, that need flexible control over the ordering and reliability of individual elements. In addition, POCv2 offers an extra feature that neither TCP nor UDP provide: a mechanism that facilitates coarse-grained[1] synchronization of multimedia elements [10].

---

[1] Coarse grained synchronization (also called temporal alignment) refers to synchronizing the start and end of elements with respect to one another. This can be distinguished from fine-grained synchronization (also called stream synchronization) which refers to keeping parallel streams synchronized with one another [8][9].

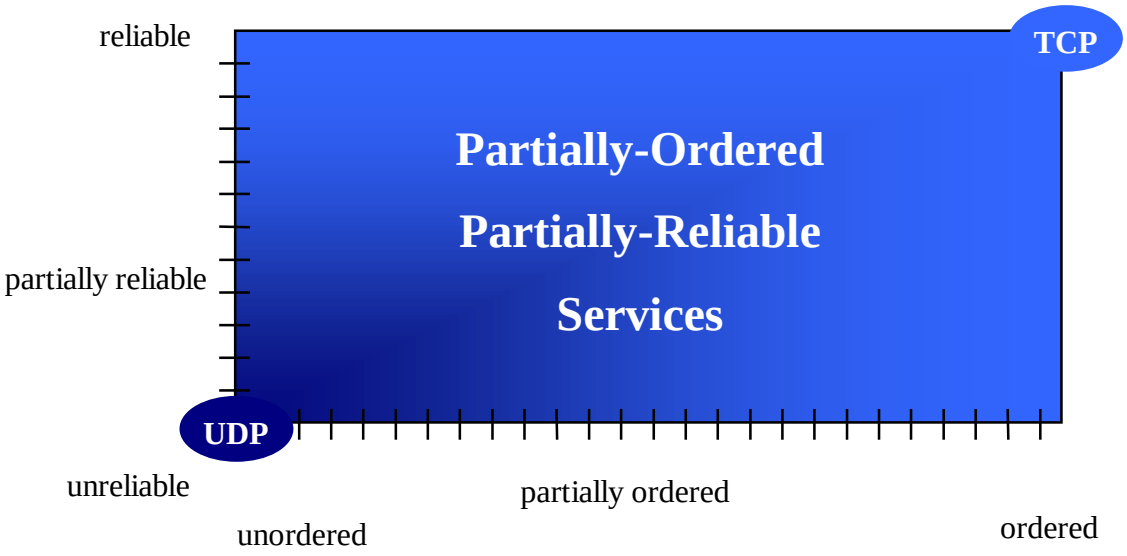


Figure 1: Spectrum of PO/PR Services

1.5 ReMDoR: POCv2's Test System

ReMDoR (Remote Multimedia Document Retrieval) is a multimedia document retrieval system that allows authors to specify synchronization requirements and varying degrees of reliability for its multimedia elements [11]. To test the benefits of POCv2, ReMDoR was developed with POCv2 features in mind. ReMDoR allows comparative testing between POCv2 and traditional protocols such as TCP and UDP.

ReMDoR's contribution to the field of Computer Science might seem questionable since many systems now exist that allow authors to construct pre-orchestrated multimedia documents. However, these existing systems are not designed to transmit multimedia documents over the Internet and present them as they

arrive. ReMDoR is one of the first systems to transmit and present multimedia documents over the Internet.

Anytime an application transmits and receives data over the Internet, consideration must be given to the case of network errors. For Internet-oriented multimedia applications, presenting the document correctly may become a more serious problem as network conditions worsen. The PEL research group has proposed that in such situations, it is appropriate to provide for "graceful degradation" of the multimedia document presentation [6]. This approach recognizes that in most multimedia documents, not all elements have equal importance or the same QoS requirements; some elements are essential to the presentation, while others are "nice to have". To provide the reliability and partial order (PO) requirements necessary, ReMDoR incorporates reliability and ordering of the individual elements directly into the authoring of the documents; thus, the authoring stage can exploit POCv2 features to insure the best possible document presentation regardless of network conditions.

ReMDoR's client/server system is being rebuilt from the "ground up" to improve performance, efficiency, and future research and development capabilities. Much of my work has contributed to the redesigning and rebuilding of ReMDoR, which is now nearly complete. The completely rebuilt system will in essence be the second generation of ReMDoR, thus earning the name ReMDoR 2.0.[2]

---

[2] All second generation system components that make up ReMDoR 2.0, such as the server, browser, and PMSL language, are also labeled version 2.0. Likewise, any reference to the first generation of ReMDoR or any component thereof will be referred to as version 1.0.

The remainder of the thesis is structured as follows. Chapter 2 explains the details of ReMDoR to give an understanding of what the system is and how it works. Additions and modifications introduced in version 2.0 are presented. Conclusions and future work are discussed in Chapter 3.

Chapter 2

ReMDoR

2.1 Architecture

The architecture of ReMDoR is depicted in Figure 2. ReMDoR consists of a client browser and a PMTP (Prototype Multimedia Transport Protocol) server that communicates over the Internet with similar interactions to that of Web browsers and HTTP servers[3]. The user specifies the URL of a document to the browser, which in turn initiates an Internet connection to the appropriate server. Once the connection is established, the browser requests the document from the server, and the server replies by sending the data for the requested document back to the browser. The browser processes the incoming data as it arrives, thereby presenting the multimedia document to the user.

The browser, however, is not responsible for delivery order and reliability requirements of the document; in fact, the browser has no knowledge about such requirements. The server communicates these requirements to the POCv2 transport layer below, and from then on, the transport layers on both sides of the connection are responsible for delivering the appropriate order and reliability requirements. If ReMDoR is running over a traditional transport protocol, such as UDP or TCP, then order and reliability requirements that are requested by the server are ignored; the

[3] Table 1 at the end of the chapter outlines the division of work among the various components of ReMDoR 2.0.

transport protocol will simply deliver the QoS that it is designed to provide: unordered/unreliable (UDP), or ordered/reliable (TCP).

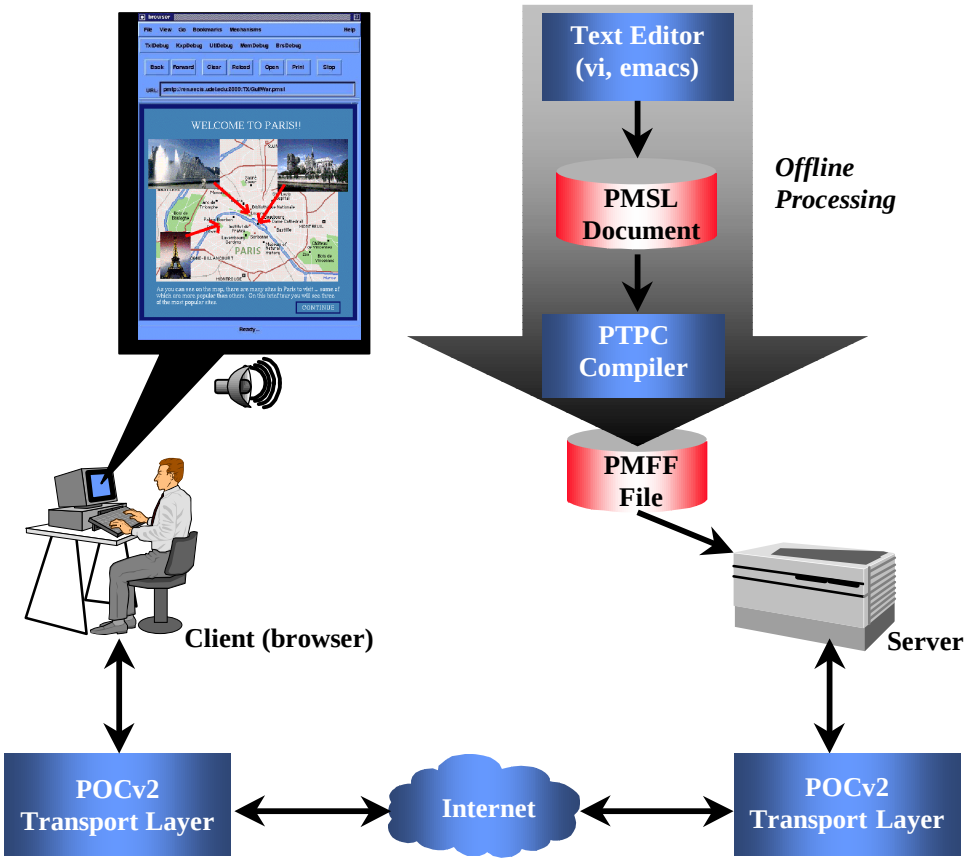


Figure 2: Architecture of ReMDoR 2.0

During the development of ReMDoR 2.0, the possibility of experimenting over more than just UDP, TCP, and POCv2 arose. The PEL research group decided to begin developing other innovative transport protocols in addition to POCv2. This posed a problem; any time a new transport protocol would be introduced, ReMDoR needed to add complex special case code to incorporate the different protocols'

Application Programming Interfaces (APIs). To avoid adding code for new APIs every time a transport protocol is developed, a Universal Transport Library (UTL) was developed. With UTL, ReMDoR can simply use a common API wrapper to access the various transport protocols available [11]. UTL puts ReMDoR in a position to serve as an experimental application for all new transport layers that are offered by UTL.

Although the basic model of interaction is similar to that used by the Web, there are many differences. The next few sections will discuss these differences by going into greater detail about the type of documents, the document specification process, the server, and the client or browser.

2.2 Temporal Documents

Unlike static Web documents, ReMDoR documents are temporal; that is, they play out over time. The documents are made of elements such as audio, still-images, text, geometric shapes, pauses, interactions, and erase events (which remove elements from the screen). During the authoring process, each element is assigned a list of successors that may consist of any number of elements (it may even be empty). Once the document is complete, there exists a logical order in which the multimedia elements may be presented. This order can be represented as a directed acyclic graph or a PO graph of all the elements, as shown in Figure 3.

2.3 Authoring ReMDoR Documents

There are two basic steps in authoring a ReMDoR document: (1) specify the document, and (2) compile the document. The document is specified using the Prototype Multimedia Specification Language (PMSL), which is presented in section

2.3.1. Once PMSL specification exists, it is compiled into the Prototype Multimedia File Format (PMFF), which is the file format in which a server actually stores the documents. The details of the compiler and the compilation process are explained in section 2.3.2. Section 2.3.3 explains the purpose of the PMFF file and how it is generated.

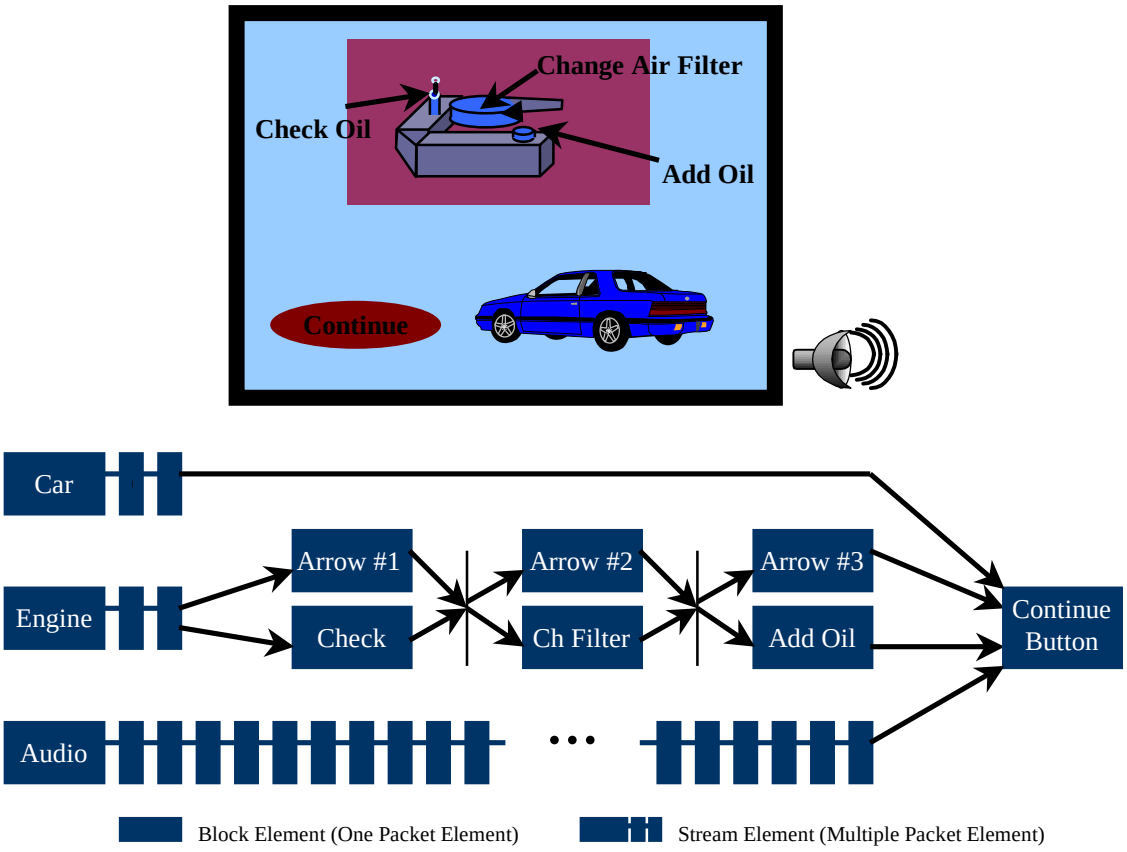


Figure 3: Example of PO Graph for a Multimedia Presentation

2.3.1 Prototype Multimedia Specification Language (PMSL)

The present version of PMSL is the second version of the language. PMSL 1.0 was developed by Phillip Conrad and Edward Golden in the PEL research group. The PMSL syntax has been modified and new features have been added to the language to ease document authoring and enhance system efficiency. Appendix A presents the legal syntax of PMSL 2.0, which is backward compatible to allow the use of existing documents already written in PMSL 1.0. The new features of PMSL 2.0 are represented in Appendix A, but will be explained more thoroughly as they come up in the next few paragraphs.

A PMSL document specification is an ASCII description of a multimedia document. PMSL documents consist of color definitions (colordefs), font definitions (fontdefs), pen definitions (pendefs), text formatting definitions (txtformatdefs), and multimedia elements (elements). Colordefs define and assign nicknames to colors that can be used later in the document. They are optional as long as none of the elements in the document need color settings. Fontdefs are similar to colordefs except that they define fonts. Fontdefs are also optional unless a text element exists. Pendefs are pre-structured combinations of variables (colors, fonts, and line widths) used by some graphic elements to specify their attributes. Pendefs are optional in two cases: (1) if the document does not contain any graphic elements that rely on these variables, and (2) if documents are written in PMSL 1.0. Txtformatdefs are also pre-structured combinations of variables that have been added in PMSL 2.0. Txtformatdefs allow text formatting to be specified in text elements by simply referring to pre-set variables; these variables are only available in PMSL 2.0 (left margin, right margin, top margin, line spacing, and justification). Txtformatdefs are completely optional, allowing an author the choice of either setting text formatting variables manually in

each text element definition, or setting them by referring to a txtformatdef. Pendefs and txtformatdefs speed up authoring documents which repeatedly use a set of variables in multiple elements; time is saved by setting the variables once in a structure definition, and then referring to the structure in multiple elements.

Elements are the multimedia elements which comprise the presentation itself. There are two kinds of elements: data elements, and control elements. Data elements present either audio or visual data to the presentation; these type of elements include: audio clips, still-images, text, and geometric shapes. Control elements, however, do not present anything; they instead control the presentation and flow of the data elements. Control elements include: pauses, interactions, and erase events (which remove objects from the screen). Since the documents are temporal, the elements require synchronization. Therefore, elements are organized into a PO which defines the presentation synchronization and ordering requirements. In addition, PMSL allows an author to define a reliability class for each element.

Other than pendefs and txtformatdefs, two more features have been added in PMSL 2.0. In PMSL 1.0, text was awkward to specify. Each line of text had to be placed in a separate element with redundant color, font, and coordinates. It was even more cumbersome to make modifications to the text, since any addition or deletion of character(s) had an affect on all of the text elements with which it was grouped; each one had to be manually modified to reflect the changes in what might be only one element in the group. In PMSL 2.0, the author specifies the margins, justification type, line spacing, and the text; the text is then automatically wrapped and formatted appropriately by the system.

The last modification made to PMSL is the incorporation of characters, underscores, and hyphens into element ID's. PMSL 1.0 only allowed numbers in the ID's, which made it confusing and awkward to create and modify documents. With the previous labeling scheme, an author had two choices: (1) try putting some logic into the numbered ID's or (2) number the ID's arbitrarily. Of course, (1) would be a wiser decision. However, after multiple revisions to a document containing many elements, it would be difficult to retain the logic in the numbered ID's assigned. In the end, most documents would inevitably end up with choice (2): ID's arbitrarily numbered. The new ID labeling scheme presented with PMSL 2.0 portrays an idea of what the element does, without having to look into the details of the element definition itself; this helps to make the source of a document more human readable.

2.3.2 PMSL To PMFF Compiler (PTPC)

After a multimedia document is specified in PMSL, one more step remains to have the document ready for retrieval over the Internet. The document must be compiled into PMFF form. The server does not fetch the PMSL specification of the document; the server expects to find a compiled version of the document (which is in PMFF form).

ReMDoR 1.0 did not compile the PMSL specification into the PMFF format. In fact, no off-line processing was performed at all. The server would fetch the PMSL document, and perform the duty of the PTPC compiler in real time as a document was being fetched. As a result, the server response time was delayed significantly, worsening with larger and more complicated PMSL documents. To the user at the browser side, the application seemed to "hang up" for a while before the document was presented. This "hang up" was due to the slow server response time.

To speed up server response time, all processing of a document specification is done offline and in advance. With this approach, when a document is requested, the server can just fetch the data it needs without any processing, and then transmit the data to the browser immediately. Basically, the server should not be "too smart"; it should be told what to do and when to do it. This new philosophy motivated the idea for the PTPC compiler and the PMFF format (explained in further detail in section 2.3.3).

The duties of the PTPC compiler are as follows. First, the PTPC compiler uses Lex and Yacc tools to parse the PMSL specification, extract all of the important data, and place the data into meaningful structures. Next, a PO graph of the document's elements is derived by traversing the entire element list and setting all of the successor pointers correctly. Once the PO graph is produced, the graph must be transitively reduced. A directed acyclic graph (PO graph) is transitively reduced if and only if for any edge, $e(v_1,v_2)$, there is no other path joining v_1 to v_2 through other vertices. Figure 4 presents an example of an original PO graph and its transitively reduced PO graph.

The next step is to set up and process the front queue from the transitively reduced PO graph (Figures 6a-h illustrate the process). The front queue is a queue of all the elements in the front of the PO graph (i.e., elements that are ready to be transmitted). Elements are fed to the front queue when the number of their predecessors that are waiting to be transmitted reaches zero. An element is removed from the front queue when all its data is transmitted. Most elements are block elements; all of the data associated with a block element is transmitted in one packet. Elements such as images and audio, on the other hand, are known as stream elements.

They require multiple transmissions to completely send all of the cells that make up the entire data. Since the compiler is not actually transmitting data, the elements are not being transmitted as they are removed from the front queue. Instead, the data is written to a file in PMFF form during PMFF generation, which happens to be the next and final stage of PTPC compilation. The PMFF format is explained in greater detail in the next section.

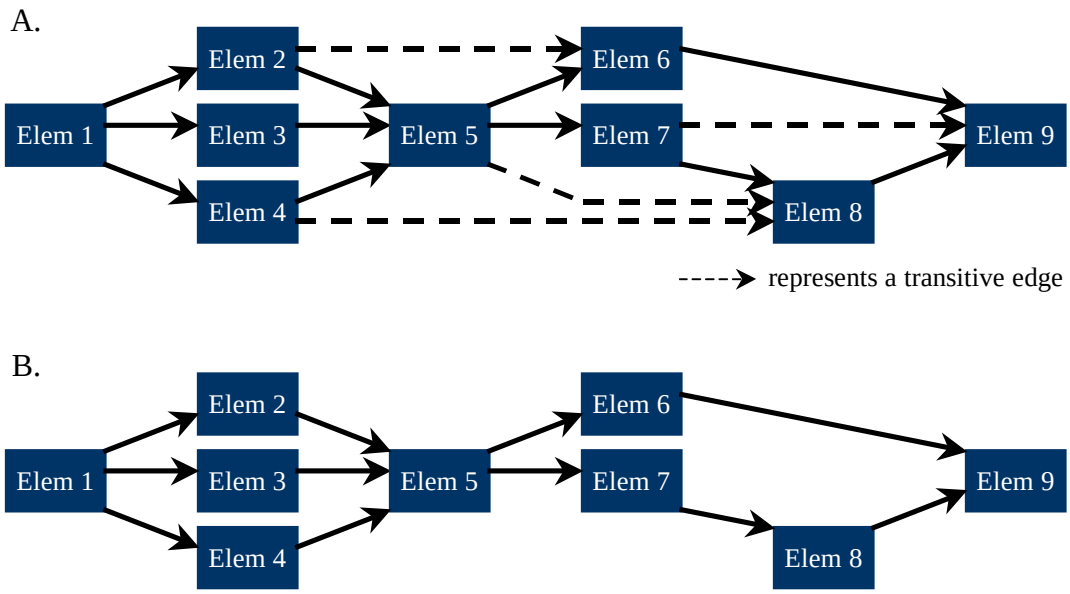


Figure 4: (A) Original PO Graph (B) Transitively Reduced PO Graph

2.3.3 Prototype Multimedia File Format (PMFF)

The format of a PMFF file is presented in Appendix B. A PMFF file is divided into three major sections: pendefs, service profile, and elements. The pendef section lists all the pens that were defined in the PMSL specification either explicitly

or implicitly (PMSL 1.0 syntax implicitly defines pens). The service profile consists of an array of integers that represent the reliability and order associated with each and every document element. Finally, the element section is the output of the front queue processing mentioned above in section 2.3.2. In essence, it is a list of "ready to go", packaged up data that the server can just read and transmit with little intelligence needed on the server's part.

2.4 Server

The duty of the server is to listen for any connection requests from browsers fetching documents that exist on the server site. ReMDoR 2.0's server is no longer aware of PMSL specifications as it once was in version 1.0; instead, the server expects to find the PMFF file for the document being requested. If the appropriate PMFF file is not found by the server, the browser is notified of the non-existing document. Otherwise, the server begins reading and transmitting the document data to the browser.

PMFF data transmission consists of three major stages. The server first transmits all the pendefs reliably, but unordered. Pendefs are transmitted first to ensure that all of the resources needed for proper document presentation arrive before any element data arrives, thus avoiding "ungraceful degradation" to the presentation. Transmitting pendefs while element data is also being transmitted may cause unnecessary delays during presentation due to loading arriving pen information. However, the pendefs use an unordered service because this service provides an advantage; if a data packet is lost on its original transmission, the client can proceed with processing the others while the lost one is retransmitted [1].

In the second stage of PMFF data transmission, the service profile is passed down to the transport layer. Without the entire service profile at the client side, elements can not receive the proper ordering and reliability they require. The server and browser, however, have no knowledge of order and reliability requirements; the transport layers on both sides of the connection are responsible for these requirements.

Element data is then transmitted in the third and final stage of PMFF data transmission. The arrangement of the element data in the PMFF file eases the job of the server. Block elements and stream elements' cells are already listed in the correct order for transmission. In addition, the POCv2 transport layer at this point already has the reliability and ordering requirements for all the elements. The server does not need to concern itself with these details; it just transmits the data in the order specified in the PMSL file, and leaves the work to the transport layer.

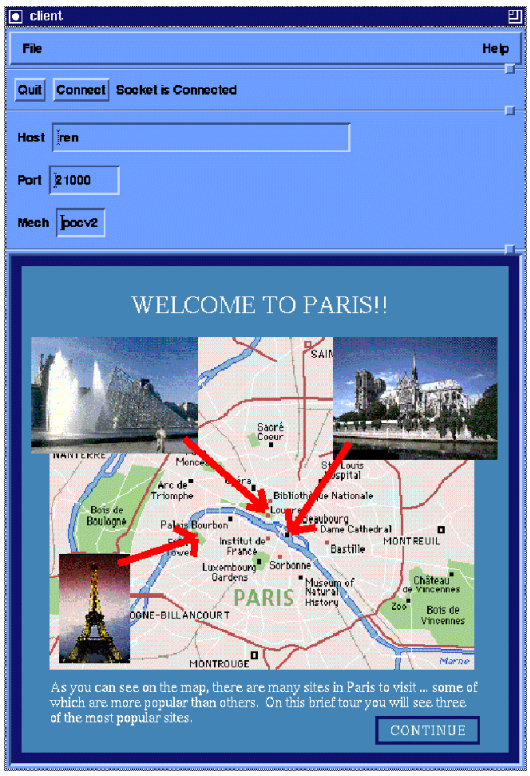
2.5 Browser (Client)

The browser allows a user to connect to a server and request a document for presentation. The browser then presents the document elements as they are delivered by the transport service [1]. The browser consists of presentation display code and user interface code. The entire browser code is much more modularized than it was in version 1.0. Improved modularization of the code allows for faster prototyping and easier integration of future research work (such as new image compressions). Since the presentation display code is presented in Conrad et al [1], only modifications will be explained. Details of the user interface will be presented due to the significant changes in ReMDoR 2.0.

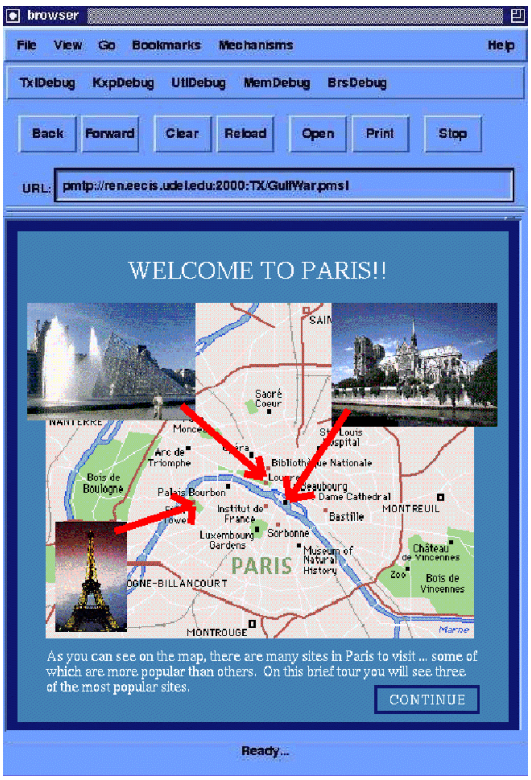
Currently, the only modification to the presentation code is in the image display process. In ReMDoR 1.0, the browser did not display any part of an image until it could be displayed in its entirety. In other words, the browser buffered image data until all of it arrived; then, it displayed the image all at once. Although the code for this type of image display processing is simple, there is a more useful way: progressive image display. This means that once the PO restraints dictate that an image is ready to be presented, the image is updated to the display as the data arrives [12]. To explain why progressive image display is preferred, the underlying goal of ReMDoR and POCv2 must be kept in mind: that goal is to deliver a faster, more "gracefully degradable" presentation to the user. System performance is enhanced when image data can be updated to the screen at an earlier point in time. With this in mind, progressive image display is clearly better.

ReMDoR 2.0 involved a major over-haul to ReMDoR 1.0's user interface (see Figure 5). The changes to the user interface create a more user-friendly environment for experimenting and data gathering. The most visible change is the Web browser look-and-feel. ReMDoR 2.0's interface incorporates: (1) a URL style of retrieving remote documents, (2) an ability to browse directories, (3) bookmarks, (4) navigation buttons, (5) a "view source" feature, and (6) a message field showing connection and presentation progress. Thanks to URL style of addressing remote documents, experiments can be run "unattended" by using an automated script. Debugging menus have been added to the interface to easily isolate various modules into producing debugging output. As a result, identifying and locating software bugs can be done quickly and easily; thus, promoting faster prototyping. With the new

user interface, the ability to run experiments via script control not only adds convenience, but is also more productive.



ReMDoR 1.0



ReMDoR 2.0

Figure 5: Changes in Browser's User Interface

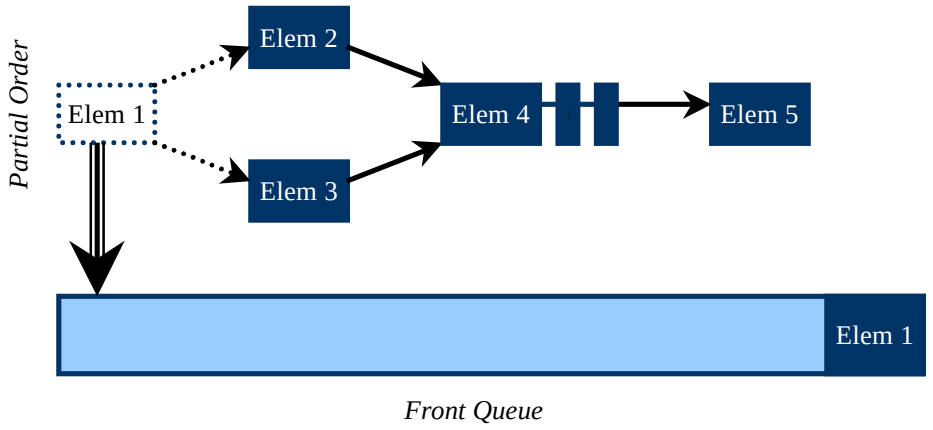


Figure 6a: (Snapshot 1) Elem 1 has no predecessors -- it is inserted into the *Front Queue*

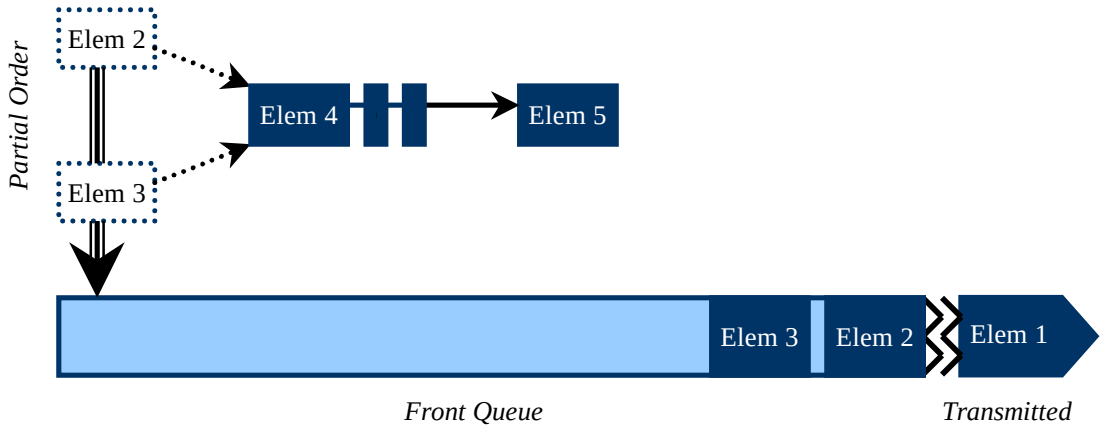


Figure 6b: (Snapshot 2) Elem 1 is de-queued & transmitted -- its successors are released; Elem 2, 3 have no predecessors -- they are inserted into the *Front Queue*

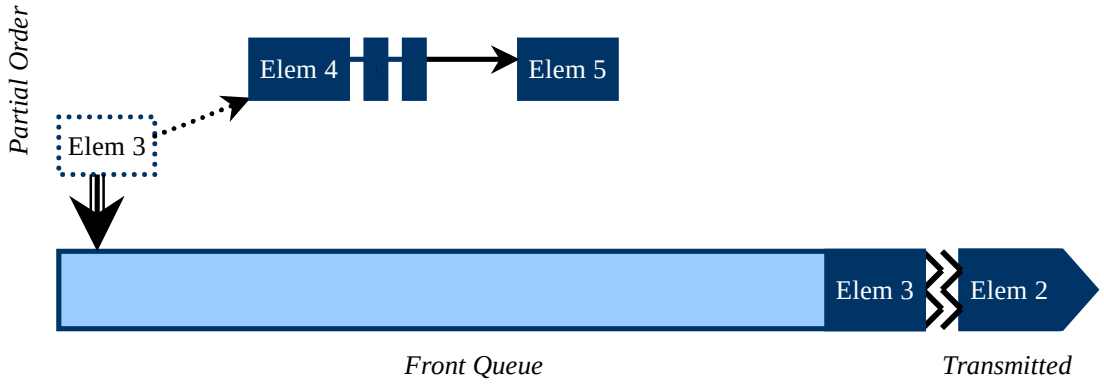


Figure 6c: (Snapshot 3) Elem 2 is de-queued & transmitted – its successors are released; Elem 3 remains in the *Front Queue;* No other element can be inserted into the *Front Queue* at this time

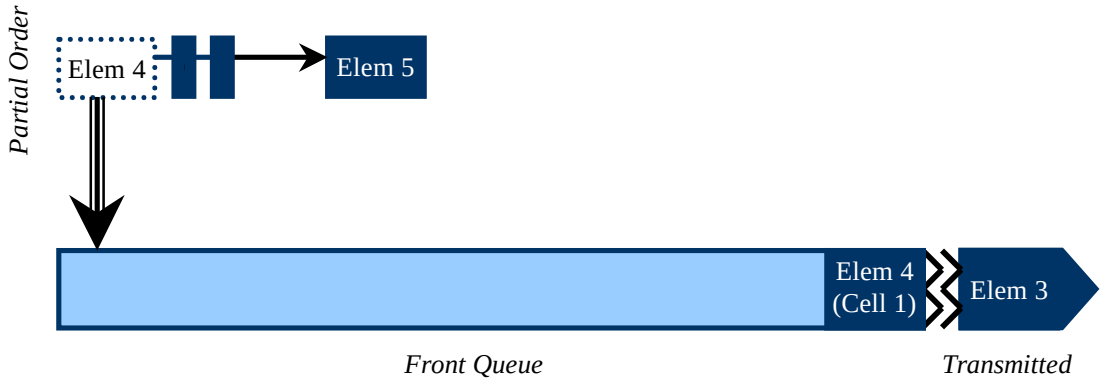


Figure 6d: (Snapshot 4) Elem 3 is de-queued & transmitted -- its successors are released; Elem 4 has no predecessors – since it is a stream element, its Cell 1 is inserted into the *Front Queue*

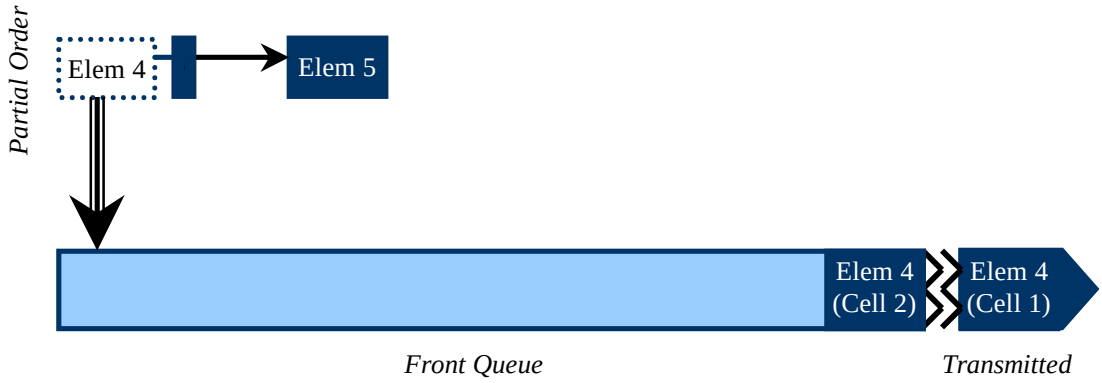


Figure 6e: (Snapshot 5) Elem 4's Cell 1 is de-queued & transmitted -- its successor can not **be released at this time; Elem 4's Cell 2 is inserted into the *Front Queue***

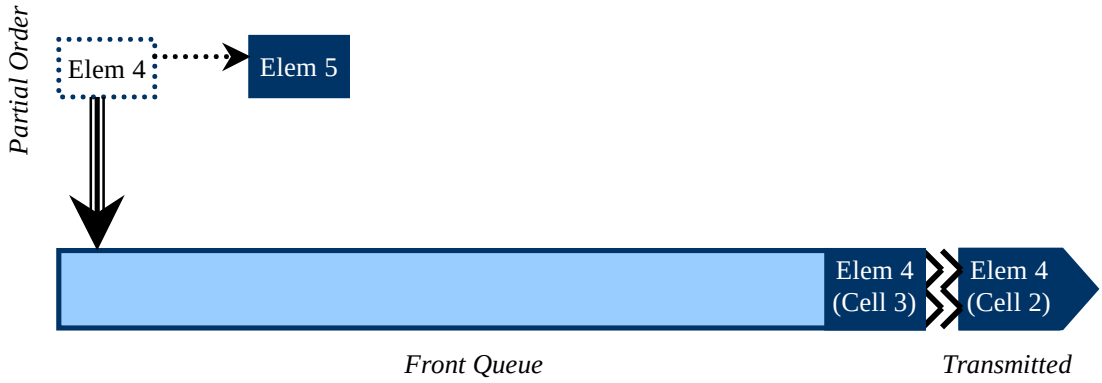


Figure 6f: (Snapshot 6) Elem 4's Cell 2 is de-queued & transmitted – its successor can not **be released at this time; Elem 4's Cell 3 is inserted into the *Front Queue***

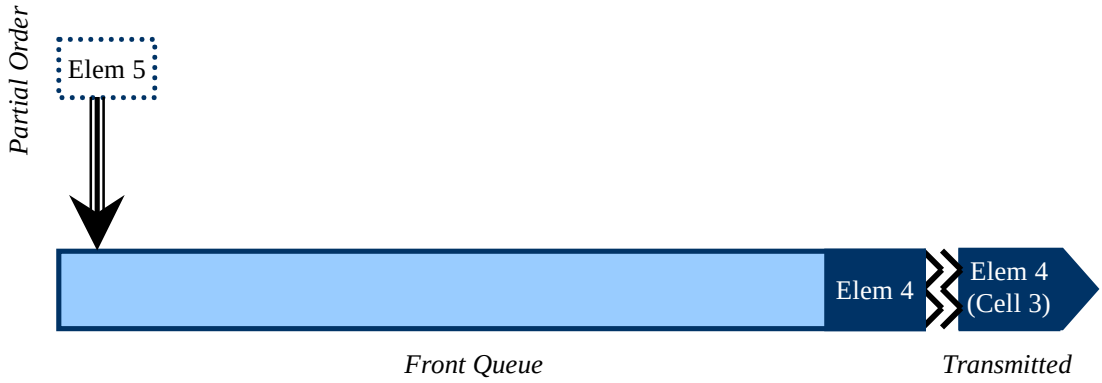


Figure 6g: (Snapshot 7) Elem 4's Cell 3 is de-queued & transmitted -- its successor can now **be released; Elem 5 is inserted into the *Front Queue***

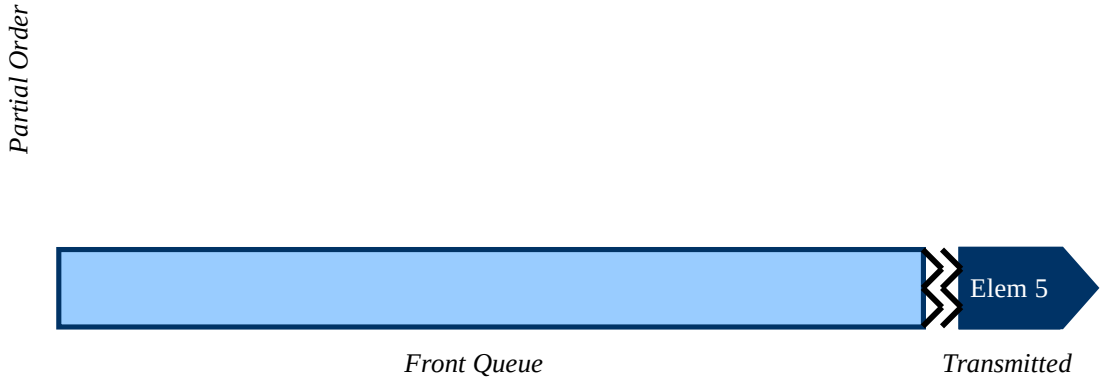


Figure 6h: (Snapshot 8) Elem 5 is de-queued & transmitted – it has no successors to release; *Front Queue* processing is done

Table 1: Division of Work

PMSL 2.0 features	
new text handling	*myself*
txtformatdefs	*myself*
new element ID's	*myself*
all other changes	*Conrad and myself*
PTPC routines	
parsing	*myself*
produce partial order	*myself*
transitive reduction	*Conrad*
PMFF generation	*myself*
PMFF	
specification	*Conrad and myself*
implementation	*myself*
Server	
specification	*Conrad and myself*
implementation	*Conrad and myself*
Browser features	
modularization	*Conrad and myself*
progressive image display	*Conrad and myself*
new user-interface	*myself*

Chapter 3

CONCLUSION AND FUTURE WORK

ReMDoR 2.0 introduces improvements that contribute to ongoing research in multimedia document retrieval over PO/PR transport protocols. First, version 1.0's instantaneous image display has been replaced by progressive image display. As a result, ReMDoR 2.0 is able to display more image data at an earlier point in time; thus, delivering a faster, more "gracefully degradable" presentation to the user. Second, modularization of code and better debugging tools allow faster software development in ReMDoR 2.0; this eases the incorporation of new features (such as video) in the system that might be useful for future research. An interface more similar to Web browsers is one of the new features that makes running experiments more convenient and more productive with ReMDoR 2.0. Additionally, the server's offline PTPC compilation enhances efficiency and speeds up document retrieval, making individual document experimenting less time consuming. Gathering data, however, requires many experiments to be repeated, which may become tedious and time consuming if they must be performed manually. To address this issue, a new scripting ability to automate experiments has been added; it improves experimenting and data gathering efficiency by allowing experiments to be repeated many times, often at night during low traffic loads, without human intervention.

The improvements in ReMDoR 2.0 contribute to the research of new network transport protocols and new image compressions. Usage of UTL, allows ReMDoR to easily incorporate new innovative protocols, such as UC, SP, and TX

[11], into the comparative testing of transport protocols. In addition, new image formats, such as Wavelet-based encoding and Network-Conscious GIF [13][14], can be easily introduced in to the newly modularized image processing routines.

ReMDoR 2.0 opens new paths and sparks new ideas for further research. With ReMDoR 2.0, POCv2 can be used to experiment with and test the performance of multimedia document retrieval over a PO/PR transport protocol. Using the Lossy Router (LR), various levels of loss can be introduced into a virtually lossless Ethernet connection between the server and the browser [6] (see Figure 7). Obviously, a PO/PR service provides no benefit if the network has no loss or reordering; however, it will be interesting to see at which loss rate PO/PR service starts to provide significant benefit that can be perceived by the user. In addition, POCv2's feature, "graceful degradation", can be put to the test over a spectrum of different network conditions.

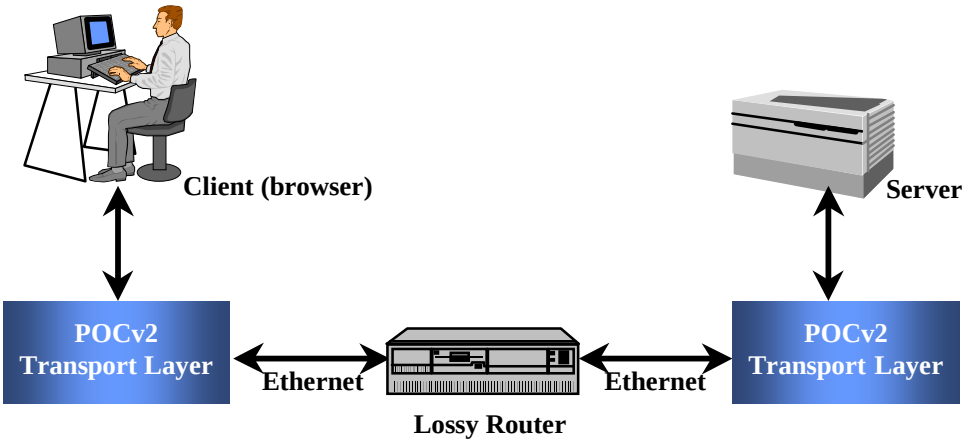


Figure 7: Testing Using the Lossy Router

ReMDoR can also be used to compare the overhead of a PO/PR service (POCv2) vs. an unordered/unreliable and vs. an ordered/reliable service. This comparison can be done by retrieving documents over POCv2 with all elements set to "no order, everything unreliable" or "total order, everything reliable", respectively; then retrieving the same documents over an unordered/unreliable and an ordered/reliable transport protocol.

Although a PO/PR service seems to be an efficient solution for remote multimedia document retrieval and other Internet applications, the appropriate definition and implementation of such a service can be quite challenging. POCv2 is only one flavor of a PO/PR service. Many researchers [15-23], including the PEL research group[24], are currently working on other definitions and implementations of PO/PR services. These efforts are working towards the goal of determining if a flexible, general transport protocol is feasible; and if so, what are the specifications of such a protocol? Incorporating new experimental transport protocols into the framework of ReMDoR as they are developed, will allow ReMDoR to be one of the vehicles used to investigate this research goal.

Appendix A

PMSL 2.0 SYNTAX

< >	non-terminals
bold	case sensitive terminals
bold & italics	case insensitive terminals
::=	equals or reduces to
|	or
~	not (everything EXCEPT what follows it)
( )	group together
[ ]	optional
{ }	zero or more occurrences

<Document> ::= [<ColordefList>] [<FontdefList>] [<PendefList>] [<TxtformatdefList>] <Elements>

<ColordefList> ::= <Colordef> {<Colordef>}
<Colordef> ::= ***COLORDEF*** <XColorName> <Id>
<XColorName> ::= <String>

<FontdefList> ::= <Fontdef> {<Fontdef>}
<Fontdef> ::= ***FONTDEF*** <Font> <Id>
<Font> ::= **" –** (<Letter> | <Digit> | **-** | ***)** {<Letter> | <Digit> | **-** | *****} **"**

<PendefList> ::= <Pendef> {<Pendef>}
<Pendef> ::= ***PENDEF*** <Id> <PendefBody>
<PendefBody> ::= <PenAttribute> {<PenAttribute>}
<PenAttribute> ::= (***FOREGROUND*** <Id>) | (***LINEWIDTH*** <Int>) | (***FONT*** <Id>)

<TxtformatdefList> ::= [<TxtformatdefList>] <Txtformatdef>
<Txtformatdef> ::= ***TXTFORMATDEF*** <Id> <TxtformatdefBody>

<TxtformatdefBody> ::= ***TOP*** <Int> ***LEFT*** <Int> ***RIGHT*** <Int> ***JUST*** <JustType> ***SPACE*** <Int>

<JustType> ::= ***L*** | ***C*** | ***R***

<Elements> ::= [<Elements>] <ElemBlock>

<ElemBlock> ::= <ElemBlockWithNext> | <ElemBlockWithoutNext>

<ElemBlockWithNext> ::= ***ELEMENT*** <Id> **:** <ElemNext> **.** <Reliability> <ElemBody>

<ElemBlockWithoutNext> ::= ***ELEMENT*** <Id> <Reliability> <ElemBody>

<Reliability> ::= ***RELIABLE*** | ***UNRELIABLE*** | ***PARTIALLY-RELIABLE***

<ElemNext> ::= [<ElemNext> **,**] <Id>

<ElemBody> ::= <Box> | <Line> | <Text> | <Audio> | <Image> | <Hotspot> | <Pause> | <Null>
| <Erase> | <End>

<Box> ::= <BoxOldSyntax> | <BoxNewSyntax>

<BoxOldSyntax> ::= ***GRAPHIC FOREGROUND*** <Id> ***LINEWIDTH*** <Int> ***. BOX*** <CornerX1>
<CornerY1> <CornerX2> <CornerY2>

<BoxNewSyntax> ::= ***PEN*** <Id> ***BOX*** <CornerX1> <CornerY1> <CornerX2> <CornerY2>

<Line> ::= <LineOldSyntax> | <LineNewSyntax>

<LineOldSyntax> ::= ***GRAPHIC FOREGROUND*** <Id> ***LINEWIDTH*** <Int> ***. LINE*** <X1> <Y1>
<X2> <Y2>

<LineNewSyntax> ::= ***PEN*** <Id> ***LINE*** <X1> <Y1> <X2> <Y2>

<X1> ::= <Y1> ::= <X2> ::= <Y2> ::= <Int>

<Text> ::= <TextOldSyntax> | <TextNewSyntaxWithFormat> | <TextNewSytaxWithoutFormat>

<TextOldSyntax> ::= ***GRAPHIC FOREGROUND*** <Id> ***FONT*** <Id> ***. TEXT*** <CornerX> <CornerY>
<String> .

<TextNewSyntaxWithOutFormat> ::= ***PEN*** <Id> ***TEXT TOP*** <Int> ***LEFT*** <Int> ***RIGHT*** <Int> ***JUST***
<JustType> ***SPACE*** <Int> <StringList>

<TextNewSyntaxWithFormat> ::= ***PEN*** <Id> ***TEXT TXTFORMAT*** <Id> <StringList>

<StringList> ::= [<StringList>] <String>

<Audio> ::= ***AUDIO*** <Path>

<Image> ::= <ImageOldSyntax> | <ImageNewSyntax>

<ImageOldSyntax> ::= ***GRAPHIC . IMAGE*** <CornerX1> <CornerY1> <Path>

<ImageNewSyntax> ::= ***IMAGE*** <ImageType> <CornerX1> <CornerY1> <Path>

<ImageType> ::= ***GIF*** | ***NCG***

<Hotspot> ::= <ContinueOldSyntax> | <PenContinue> | <PenLink> | <NoPenContinue> | <NoPenLink>

<ContinueOldSyntax> ::= ***GRAPHIC . HOTSPOT*** <CornerX1> <CornerY1> <CornerX2> <CornerY2> ***CONTINUE***

<PenContinue> ::= ***PEN*** <Id> ***HOTSPOT*** <CornerX1> <CornerY1> <CornerX2> <CornerY2> ***CONTINUE***

<PenLink> ::= ***PEN*** <Id> ***HOTSPOT*** <CornerX1> <CornerY1> <CornerX2> <CornerY2> ***LINK*** <Url>

<NoPenContinue> ::= ***HOTSPOT*** <CornerX1> <CornerY1> <CornerX2> <CornerY2> ***CONTINUE***

<NoPenLink> ::= ***HOTSPOT*** <CornerX1> <CornerY1> <CornerX2> <CornerY2> ***LINK*** <Url>

<Url> ::= **"pmtp://** <Hostname> **:** <Port> **:** <Mechanism> **/** <Path> **"**

<Hostname> ::= <HostnameSegment> **.** {<HostnameSegment> **.**} <HostnameSegment>

<HostnameSegment> ::= <Letter> {<Letter>}

<Port> ::= <Int>

<Path> ::= [/] <PathSegment> {/ [<PathSegment>]} [/]

<PathSegment> ::= (<Letter> | <Digit> | **.** | **#** | **_** | **-**) { <Letter> | <Digit> | **.** | **#** | **_** | **-** }

<Pause> ::= ***PAUSE*** <Time>

<Time> ::= <Int>

<Null> ::= ***NULL***

<Erase> ::= ***ERASE*** <Id>

<End> ::= ***END***

<Letter> ::= ***A*** | ***B*** | ***C*** | ***D*** | ***E*** | ***F*** | ***G*** | ***H*** | ***I*** | ***J*** | ***K*** | ***L*** | ***M*** | ***N*** | ***O*** | ***P*** | ***Q*** | ***R*** | ***S*** | ***T*** | ***U*** | ***V*** | ***W*** | ***X*** | ***Y*** | ***Z***

<Digit> ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

<Int> ::= <Digit> {<Digit>}

<CharId> ::= <Letter> {<Letter> | <Digit> | **_** | **-**}

<Id> ::= <Int> | <CharId>

<CornerX1> ::= <CornerY1> ::= <CornerX2> ::= <CornerY2> ::= <Int>

<String> ::= " {~(**\n**)} "

<FilePathName> ::= <String>

Appendix B

PMFF FILE STRUCTURE

< >	non-terminals
bold	terminals
::=	equals or reduces to
|	or
~	not (everything EXCEPT what follows it)
( )	group together
[ ]	optional
{ }	zero or more occurrences
% italics	everything past *%* is a comment

<File> ::= <PendefSection> <ServiceProfile> <ElementSection>

<PendefSection> ::= <NumPens> {<Pen>}

<NumPens> ::= <Int> *% total number of pens in the pendefs section*

<Pen> ::= <PenBodyLength> <PenBody> **\n**

<PenBodyLength> ::= <Int> *% total number of characters (including spaces) in the body of the pen*

<PenBody> :: = **pen** <PenNum> / <NumPens-1> <PenId> **fg** <Color> **lw** <Linewidth> **ft** <Font>

<PenNum> ::= <Int> *% the number assigned to this pen (numbered 0 to <NumPens> - 1)*

<NumPens-1> ::= <Int> *% total number of pens minus one (<NumPens> - 1)*

<PenId> :: = (<Int> | <CharId>) | <PenOnTheFlyId>

<PenOnTheFlyId> ::= **_$PEN** <Int> **_**

<Çolor> ::= <String> | **0** *% a 0 means the color is not specified*

<Linewidth> ::= <Int> *% a linewidth of -1 means it is not specified*

<Font> ::= (**"** – (<Letter> | <Digit> | **-** | *****) {<Letter> | <Digit> | **-** | *****} **")** | <NoFont>

<NoFont> ::= **0** % a 0 means the font is not specified

<ServiceProfile> ::= <ServiceProfileLength> <NumElements> <ElemProfile> {ElemProfile}

<ServiceProfileLength> ::= <Int> *% number of 32-bit integers (including length) in entire profile*

<NumElements> ::= <Int> *% total number of elements in the document*

<ElemProfile> ::= <Reliability> <NumSuccessors> <Successor>

<NumSuccessors> ::= <Int> *% number of successors the element has*

<Successor> ::= <Int> *% successor's element number (implicitly numbered in the service profile)*

<ElementSection> ::= <Element> {<Element>}

<Element> ::= <ElemNum> <MoreCells?> <NumFileBytesToRead> <ElemBodyLength> **elem**
<ElemNum> / <NumElements-1> <ElemId> <ElemBody> [<ElemTrailer>] **\n**

<ElemeNum> ::= <Int> *% the number assigned to this element (numbered 0 to N-1)*

<MoreCells?> ::= <Int> *% 1 if this element has more cells (or data packets) to be sent; 0 if not*

<NumFileBytesToRead> ::= <Int> *% number of bytes that need to read from a file*

<ElemBodyLength> ::= <Int> *% number of characters (including spaces) in the body of the element*

<NumElements-1> ::= <Int> *% total number of elements minus one (<NumElements> - 1)*

<ElemId> ::= <Id>

<ElemBody> ::= <Graphic> | <Hotspot> | <Erase> | <Audio> | <Pause> | <Null> | <End>

<ElemTrailer> ::= <FileOffset> [<FilePath>] *% FilePath is only sent with the first cell*

<FilePath> ::= [/] <PathSegment> {/ [<PathSegment>]}

<PathSegment> ::= (<Letter> | <Digit> | **.** | # | _ | -) { <Letter> | <Digit> | **.** | # | _ | - }

<Graphic> ::= **draw** (<Line> | <Box> | <Text> | <Image>)

<Line> ::= **line** <X1> <Y1> <X2> <Y2>

<X1> ::= <Y1> ::= <X2> ::= <Y2> ::= <Int>

<Box> ::= **box** <CornerX1> <CornerY1> <CornerX2> <CornerY2>

<CornerX1> ::= <CornerY1> ::= <CornerX2> ::= <CornerY2> ::= <Int>

<Text> ::= **text** <TopMargin> <LeftMargin> <RightMargin> <Justification> <Spacing> **\n** <StrList>

<TopMargin> ::= <LeftMargin> ::= <RightMaring> ::= <Spacing> ::= <Int>

<Justification> ::= **l** | **c** | **r**

<StrList> ::= <String> {<String>}

<String> ::= " {~(**\n**)} "

<Image> ::= **image** <ImageType> <CornerX1> <CornerY1>
<ImageType> ::= **gif** | **ncg**

<Hotspot> ::= **hotspot** <PenNum> <CornerX1> <CornerY1> <CornerX2> <CornerY2>
<HotspotType> <Url>

<HotspotType> ::= **c** | **l** *% c = continue hotspot; l = link hotspot*

<Url> ::= <NoUrl> | (**"pmtp://** <Hostname> **:** <Port> **:** <Mechanism> / <Path> ")

<NoUrl> ::= **0**

<Hostname> ::= <HostnameSegment> **.** {<HostnameSegment> **.**} <HostnameSegment>

<HostnameSegment> ::= <Letter> {<Letter>}

<Port> ::= <Int>

<Path> ::= <FilePath> [/]

<Erase> ::= **erase** <ElemNum>

<Audio> ::= **audio**

<Pause> ::= **pause** <PauseTime>

<PauseTime> ::= <Int>

<Null> ::= **null**

<End> ::= **end**

<Letter> ::= ***A*** | ***B*** | ***C*** | ***D*** | ***E*** | ***F*** | ***G*** | ***H*** | ***I*** | ***J*** | ***K*** | ***L*** | ***M*** | ***N*** | ***O*** | ***P*** | ***Q*** | ***R*** | ***S*** | ***T*** | ***U*** | ***V*** | ***W*** | ***X*** | ***Y*** | ***Z***

<Digit> ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

<Int> ::= <Digit> {<Digit>}

<CharId> ::= <Letter> {<Letter> | <Digit> | **_** | **-**}

<Id> ::= <Int> | <CharId>

REFERENCES

[1] P. T. Conrad, E. Golden, P. D. Amer, and R. Marasli. A multimedia document retrieval system using partially-ordered/partially-reliable transport service. In *Multimedia Computing and Networking 1996 (MMCN96; sponsored by SPIE/IS&T)*, San Jose, CA, USA, January 1996.

[2] S. Iren, P. D. Amer, and P. T. Conrad. The transport layer: Tutorial and survey. Technical Report 98-02, CIS Dept., University of Delaware, January 1998.

[3] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, 1996.

[4] J. B. Postel. User Datagram Protocol. Internet Request for Comments RFC768, August 1981.

[5] J. B. Postel. Transmission Control Protocol. Internet Request for Comments RFC793, September 1981.

[6] P. D. Amer, P. T. Conrad, E. Golden, S. Iren, and A. Caro. Partially-ordered, partially-reliable transport service for multimedia applications. In *Advanced Telecommunications/Information Distribution Research Program Annual Conference*, College Park, MD, January 1997.

[7] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *SIGCOMM '90*, pages 200-209, Philadelphia, Pennsylvania, September 1990. ACM. Computer Communications Review, Vol. 20(4), September 1990.

[8] M. Wynblatt. Position statement on multimedia synchronization. In *IEEE Workshop on Multimedia Synchronization*, Tysons Corner, VA, May 1995.

[9] J. Schnepf, J. A. Konstan, and D. Du. Dong FLIPS: Flexible interactive presentation synchronization. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 213-222, Washington, DC, May 1995. IEEE.

[10] P. T. Conrad, P. D. Amer, R. Marasli. Graceful degradation of multimedia documents via partial order and partial reliability transport protocols. In *IEEE Workshop on Multimedia Synchronization*, Tysons Corner, VA, May 1995.

[11] P. T. Conrad, P. D. Amer, M. Taube, G. Sezen, S. Iren, and A. Caro. Testing environment for innovative transport protocols. In *Advanced Telecommunications/Information Distribution Research Program Annual Conference*, College Park, MD, February 1998.

[12] K. Sayood. *Introduction to Data Compression.* Morgan Kaufmann Publishers, Inc, 1996.

[13] P. D. Amer, S. Iren, G. Sezen, P. T. Conrad, M. Taube, and A. Caro. Network-conscious GIF image transmission over the Internet. In *4th International Workshop on High Performance Protocol Architectures (HIPPARCH '98)*, June 1998.

[14] S. Iren, P. D. Amer, and P. T. Conrad. Network-conscious compressed images over wireless networks. In *5th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS' 98)* Oslo, Norway, September 1998.

[15] M. Ahuja. Flush primitives for asynchronous distributed systems. *Info Processing Letters*, 34(1):5-12, February 1990.

[16] K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36-53, December 1993.

[17] D. Cheriton. VMTP: Versatile message transaction protocol specification. (Internet) Network Working Group, Request for Comments RFC1015, April 1993.

[18] B. J. Dempsey. *Retransmission-Based Error Control For Continuous Media Traffic in Packet-Switched Networks.* PhD thesis, University of Virginia, 1991.

[19] F. Gong and G. Parulkar. An application-oriented error control scheme for high-speed networks. Technical Report WUCS-92-37, Department of Computer Science, Washington University in St. Louis, November 1992.

[20] L. Lamport. Time, clocks and the ordering of events in a distributed system. *CACM*, 21(7):558-565, July 1978.

[21] G. Neiger and S. Toueg. Substituing for real time and common knowledge in asynchronous distributed systems. In *Proc 4th Symp on Principles of Distributed Computing*, pages 281-293, 1987.

[22] L. Peterson, N. Buchholz, and R. Schlighting. Preserving and using context information in interprocess communication. *ACM Trans on Computer Systems*, 7(3):217-218, August 1989.

[23] T. F. La Porta and M. Schwartz. The multistream protocol: A highly flexible high-speed transport protocol. *IEEE Journal on Selected Areas in Communications*, 11(1):519-530, May 1993.

[24] E. Golden. *TRUMP: Timed-Reliability Unordered Message Protocol*. MS Thesis, CIS Dept., University of Delaware, 1997.

[25] P. T. Conrad. *Order, Reliability, and Synchronization in Transport Layer Protocols for Multimedia Document Retrieval*. PhD Dissertation, CIS Dept., University of Delaware, (in progress).

[26] S. Iren, P. D. Amer, A. Caro, P. T. Conrad, G. Sezen, and M. Taube. Network-conscious compressed image transmission over battlefield networks. In *Advanced Telecommunications/Information Distribution Research Program Annual Conference*, College Park, MD, February 1998.

[27] P. D. Amer, C. Chassot, T. J. Connolly, M. Diaz, and P. T. Conrad. Partial order transport service for multimedia and other applications. *IEEE/ACM Trans on Networking*, 2(5):440-456, October 1994.

[28] T. Connolly, P. D. Amer, and P. T. Conrad. An extension to TCP: Partial order service. Request for Comments (Experimental) RFC 1693, Internet Engineering Task Force, November 1994.

[29] P. T. Conrad, P. D. Amer, E. Golden, S. Iren, R. Marasli, and A. Caro. Transport QoS over unreliable networks: No guarantees, no free lunch! In *IFIP Fifth International Workshop on Quality of Service (IWQOS '98)*, New York, NY, USA, May 1997.

[30] R. Marasli, P. D. Amer, and P. T. Conrad. Retransmission-based partially reliable services: An analytic model. In *IEEE INFOCOM*, San Fransisco, California, March 1996.

[31] R. Marasli, P. D. Amer, P. T. Conrad, and G. Burch. Partial order transport service: An analytic model. In *Ninth Annual IEEE Workshop on computer Communications*, Marathon, Florida, October 1991.

[32] R. Marasli. *Partially Ordered and Partially Reliable Transport Protocols: Performance Analysis.* PhD thesis, CIS Dept., University of Delaware, 1997.

[33] J. A. Rody and A. Karmouch. A remote presentation agent for multimedia databases. In *Proceedings of the International Conference on Multimedia Computing and Systems,* pages 223-230, Washington, DC, May 1995. IEEE.

[34] W. R. Stevens. *UNIX Network Programming*. Prentice-Hall, 1990.