MAEstro — A Distributed Multimedia Authoring Environment *

George D. Drapeau
Stanford University
Stanford, CA 94305-3090
drapeau@air.stanford.edu

Howard Greenfield
Sun Microsystems
Mountain View, CA
howardg@sun.com

Abstract

Current multimedia authoring environments are typically designed as self-contained systems; one application is completely responsible for the creation of multimedia documents. The number of media supported by such systems is usually small, and accommodating new media typically requires rewriting the authoring system.

Meanwhile, workstation vendors are adding multimedia support to distinguish their products from those of their competitors. As a result, new media are being introduced on the workstation platform so quickly that monolithic authoring systems incorporating today's state-of-the-art media are often obsolete within a year.

MAEstro was designed for extensibility. The key to **MAEstro** is its inter-application messaging system, similar to NeXT's Speaker-Listener protocol [1]. Through the messaging system, an authoring application controls a number of "media editors" (applications responsible for the manipulation of a particular medium or source of information). When creating a multimedia document, the authoring application asks the media editors for information about their current selections; during playback, the authoring application sends messages to the media editors telling them to perform media selections.

At present, **MAEstro** consists of a suite of applications on Sun and NeXT workstations. Media editors on the Sun support text, CD audio, videodiscs, and a text search engine. The NeXT runs a MIDI music editor. The authoring application is a timeline editor running on the Sun that allows authors to create multimedia documents using any combination of media on both workstations. For example, an author can create a multimedia document that synchronizes NeXT-controlled MIDI synthesizers with video controlled by the Sun.

1 Introduction

Multimedia authoring environments are typically centralized authoring environments, single stand-alone applications that provide built-in support for a number of media. Macromind's MediaMaker, Apple's Hypercard, and Imagine's MediaStation are examples of this category. A centralized authoring environment provides built-in support for a limited set of media, and the interface is designed specifically for those media. Since the authoring environment tries to support several media, support for any one medium is generally much weaker than that of an application devoted solely to that medium. For example, text support in Hypercard is not as powerful as that provided by Microsoft Word.

Centralized authoring environments provide single-source support of multiple media by trading away flexibility and power in any one of them. Although some environments provide hooks through which programmers can add support for new media, the user interface is not designed to handle them coherently. In addition, centralized authoring environments are designed explicitly with one model of authorship in mind and so are not flexible enough to accommodate new authorship models. Accommodating new media and new

*This project is part of a research effort between Stanford University, Academic Information Resources and Sun Microsystems' Collaborative Research and Worldwide Education and Research Marketing groups. We gratefully acknowledge Sun's support and funding of this project.

styles of authorship is necessary because change is the norm in the multimedia market and will remain so for the next few years.

Some have addressed these issues through the creation of courseware, customized software for a particular instructional or research domain. Courseware is typically written for a particular faculty member, the goal being to provide an authoring model specifically for that faculty's curriculum. The customized nature of courseware gives faculty the choice of media they want but does so on a case-by-case basis, thereby limiting the potential audience that can benefit from courseware.

In contrast to the courseware model, we have addressed the issue of wide access to media by creating **MAEstro**, a workstation-based multimedia authoring environment that focuses on authorship of multimedia documents. Our goal is to create a rich environment that is simple to use while providing support for a wide variety of media. **MAEstro** is scalable so that authors can create multimedia documents with whatever media are available to them. A student can create multimedia documents in public workstation labs that do not have direct access to "hard" media such as videodisc and compact disc players but may have text, graphics, and some audio capabilities.

The project is currently planning on two delivery sites within the Stanford campus: a small multimedia lab providing a wide variety of audio and video capabilities that is accessible to faculty and students at Stanford, and public workstation clusters that provide software-only media but none of the more exotic media supported by the multimedia lab. The two sites were chosen to test the scalability of **MAEstro** and to deliver the environment to a large community of potential authors.

Section 2 describes the issues that influenced the design of the **MAEstro** environment. Section 3 lists the components of the environment. Section 4 explains the authorship model underlying the design of **MAEstro**. Section 5 describes the inter-application messaging system designed to support the **MAEstro** model of authorship. Section 6 discusses plans to add new media and functionality to the environment. Section 7 discusses benefits and problems of the system.

2 Design Issues

New media are introduced to the workstation market every few months, making it extremely difficult to create a stand-alone authoring environment that coherently supports current multimedia products. Even if such an environment were built, new hardware and software introduced over the next year would likely make the authoring environment obsolete. Our greatest problem during the design process was dealing with uncertainty: which media should be supported? What should our authoring application look like? How should new media be accommodated? Who will be our authors? How do we work user input into future designs? How do we reduce the learning curve for new authors?

Section 2.1 discusses the factors that influenced the implementation of **MAEstro**. Section 2.2 defines the term "media" as used in the **MAEstro** environment. The **MAEstro** notion of authorship is defined in Section 2.3. Section 2.4 contrasts the **MAEstro** model of multimedia authorship with the more traditional courseware model.

2.1 Requirements

Demand for media has grown to a point where most people want to take advantage of multimedia capabilities in their own software offerings. However, different groups of people have different requirements of media; for example, a music researcher might be interested only in MIDI-controlled audio and simple graphics, while a graduate student in German Studies might want to synchronize video with digitized audio in several languages; an engineering student might want to combine the visual results of a simulation engine with a textual description of the simulation. There is no one set of media that will satisfy everyone; furthermore, some require media not yet available to us. In short, we do not know which media authors will want nor how they will combine media.

We learned from the courseware model that authorship takes many forms; in essence, courseware is all about writing a new authorship model for each faculty member. This means that an authoring environment should be designed to accommodate different styles of authorship, or different ways to structure information. To impose a single style of authorship and expect it to suit everyone's purposes is a mistake.

The workstation platform is still very much in an experimental stage of multimedia development; multimedia itself is a moving target and we expect it will remain so for the next few years. To serve the computing community well, an authoring environment should be designed for extensibility so that new media can be integrated with the existing environment as they become available. At the same time, we recognize that a limited development staff affects the ability to write tools for new media; the environment should therefore allow new media to be added without perturbing any existing code if possible. A limited staff necessitates localized changes to the environment.

The primary issues driving the current implementation of **MAEstro** are the ability to allow the author to pick and choose her set of media, the ability to accommodate diverse styles of authorship, and extensibility that is as painless as possible to application programmers. Portability is also desirable since our public workstation rooms include workstations from several vendors.

The final requirement is defined by the tools available in our Sight and Sound lab, a public access room providing video and audio hardware and software for use by all students and faculty on campus. The lab dedicates machines to particular media (for example, a NeXT computer serves as an audio workstation, while a Sun controls videodisc devices). To allow authors to create multimedia documents that include both types of media, it is desirable to create an environment that allows authorship of documents that span machines; in other words, interoperability. This requirement is different than that of portability, which allows us to take an existing environment to a different computer. Interoperability allows us to use services from multiple, heterogeneous computers simultaneously.

2.2 Defining "Multimedia"

Discussion of multimedia typically includes the media of text, images, audio, and video. Some choose to categorize these media in more specific terms; for example, material describing the Intel DVI specification treats video as three components: live motion video, video special effects, and video stills [2, 3]. Our feeling is that even these forms of media cannot necessarily be classified in such broad terms, since different people will use the same basic media in different ways. For example, a musician might use three different forms of audio: digitally sampled audio that is stored on disk, audio generated by a digital signal processor (DSP), and MIDI signal data to control external synthesizers. Software to control these three sources of audio would most likely treat them differently from one another, and yet they are still generally considered "audio". This distinction drives our definition of media.

For the **MAEstro** environment, we define media as *sources of information*. Any source of information is its own medium, and usually has an application devoted to the manipulation of that source. Another way of saying this is that *media are defined by the applications written for them.* Thus, the application responsible for controlling MIDI-based audio defines MIDI as its own medium; the digital sampled audio application defines digital audio as yet another medium.

In considering the needs of humanities faculty and students we found that although they want alternate media as part of their computing environment, these media would not be the whole of that environment. They want to integrate new media with data manipulated by their own domain-specific applications. By considering any application or source of information as a medium, the scope of the term "multimedia" widens to include not only audio and video, but also text search and retrieval engines, simulation applications, visualization applications, spreadsheets, and so on. This model offers authors a richer combination of media than a more traditional definition of media. For example, we envision a researcher using text search queries and their results as part of a multimedia document then adding the researcher's own voice annotations and text from a word processor to explain the meaning of the search. The number of media possible is limited only to the number of applications available on the workstation.

2.3 Defining "Authorship"

Authorship is the process of creating a document. The nature of the document and process by which a document is created is highly variable, but the goal of the authorship process remains to create a multimedia document.

Authorship may take any number of forms. Hypercard is an authoring application in which information is structured as a series of cards the author can link together. Macromind's Director for the Macintosh is

another authoring application, revolving around the presentation of multimedia "slide shows". A musical score editor would be another authoring application; the structure of such an application's documents is obvious to musicians. The NeXT Mail application might also be considered an authoring application. The structure of a document in NeXT mail is a series of mail messages that can be plain text or "packages" that include "attachments" (documents from other applications). Authoring a document with the NeXT Mail application means composing a package that is sent to another person.

2.4 The MAEstro Model of Authorship vs. the Courseware Model

MAEstro was designed to address some of the problems inherent in the courseware model of multimedia software development. The courseware model focuses on the structured presentation of media; usually, a student reacts to an existing document by choosing from a limited number of predefined options. The **MAEstro** model of authorship focuses on the creation of documents rather than presentation of pre-formed documents.

Stanford currently has a small group of programmers who create courseware for faculty on campus. The courseware model at Stanford typically works as follows: a faculty member decides that she wants to use the computer as part of the class curriculum. The courseware development group devotes one of the group's programmers to the faculty member for some period of time. The programmer and faculty member meet initially to discuss goals and requirements. During development, the programmer meets occasionally with the faculty member to insure the accuracy of the material written into the courseware and to get feedback from the faculty member. Development time for courseware projects varies widely; projects have lasted as little as a few months and as long as three years.

The courseware development group at Stanford has produced some software of great use to faculty, and we believe the courseware development model to be a useful form of software development. However, there are a number of problems with the courseware model:

- **Development is labor-intensive.** Writing courseware is difficult and time-consuming; as a result, faculty do very little if any development of their own. The work is almost completely done by the programmer. Lending a programmer to a faculty member for some period of time means that only a limited number of documents (courseware offerings) can be created in a given time frame. The courseware development group at Stanford has seven programmers; even if each programmer could produce two complete courseware offerings per year, the vast majority of the faculty community would be neglected.

- **Productivity/success is unpredictable.** It is difficult to predict how long a courseware project will take to complete and how useful it will be once completed. The courseware development group has spent three years developing courseware that is used by a single class for a period of perhaps two weeks. The group has also spent less than a year to produce courseware that can be used by several foreign language classes for the duration of an academic term or more. Taking on a courseware development project entails a great deal of risk.

- **Courseware is often "read-only".** For most courseware, information flows from the teacher to the student; the student is simply reacting to information structured by the teacher and programmer. In many courseware offerings, the student is interacting with a pre-made, structured document that offers yes or no choices along the way. More sophisticated courseware allows the student to set a larger number of parameters; even so, the student is still not modifying or adding to the knowledge contained in the courseware. The student cannot draw out more information from the courseware than has already been programmed in. This denies the student the learning associated with the act of creation, as happens when writing a paper. Often the programmer learns more about the subject matter by creating the courseware than the student does by using it.

MAEstro addresses the problems above by taking the approach that students are authors. The job of our programmers, then, is to provide tools and applications that make authorship simpler for students. The objective of courseware is often to automate the teaching process; by making students into their own authors, we can forget about trying to automate teaching and focus on simpler tools to help students write multimedia

"papers". This model puts more burden on the students, but our hope is that the multimedia tools will engage students enough that they will be motivated to create. We believe that focusing on students creating their own multimedia documents can potentially serve a greater segment of the computing community than the courseware model.

The resulting architecture is described in the next section.

3 System Overview

MAEstro consists of four logical components:

1. Media editors
2. An authoring application
3. An inter-application messaging system
4. The Port Manager application

Just as work in an office environment is spread among specialists, authorship in **MAEstro** is spread among applications well suited to a particular medium, called *media editors*. An authoring application serves as the "office manager", coordinating the actions of the other applications; it does so by sending messages to the applications, telling them to select pieces of a document and perform them. The authoring application communicates with the other applications via an inter-application messaging system designed for the environment. Applications advertise their services by registering themselves with an application called the Port Manager. An application can query the Port Manager to discover which other applications are currently advertising their services.

Section 3.1 defines the notion of a media editor and lists media editors in the current environment. The authoring application is described in Section 3.2. The inter-application messaging system is described in Section 3.3. Section 3.4 describes the Port Manager.

3.1 Media Editors

Recall from Section 2.2 that media are defined by the applications written for them. In **MAEstro**, these applications are called "media editors". As new media are introduced to the workstation platform, media editors are written to give authors access to and control of those media. Media editors do not generally stand alone, rather they are used to create pieces of a multimedia document. The overall structure of a complete multimedia document is managed by the authoring application.

Our current environment includes the following media editors:

- **cdEdit**— An application used by authors to annotate audio compact discs. The application stores a list of start and stop points corresponding to audio segments on the disc.
- **videoEdit**— An application for annotating videodiscs. The author creates lists of start and end points that define single frames of video or complete scenes.
- **QuoteMaker**— An application for quoting existing text, as opposed to a full word processing application. The author opens a text for browsing; selecting part of the text creates a "quote" that can be saved as part of a list of quotes. The author later displays the quote in large type on a separate window as part of a multimedia presentation.
- **Searcher**— A text search and retrieval engine application used primarily by humanities researchers as an analysis tool for online text. The application stores a list of queries the author makes of a particular text; the author can later open the list of queries and resume searching on that text.

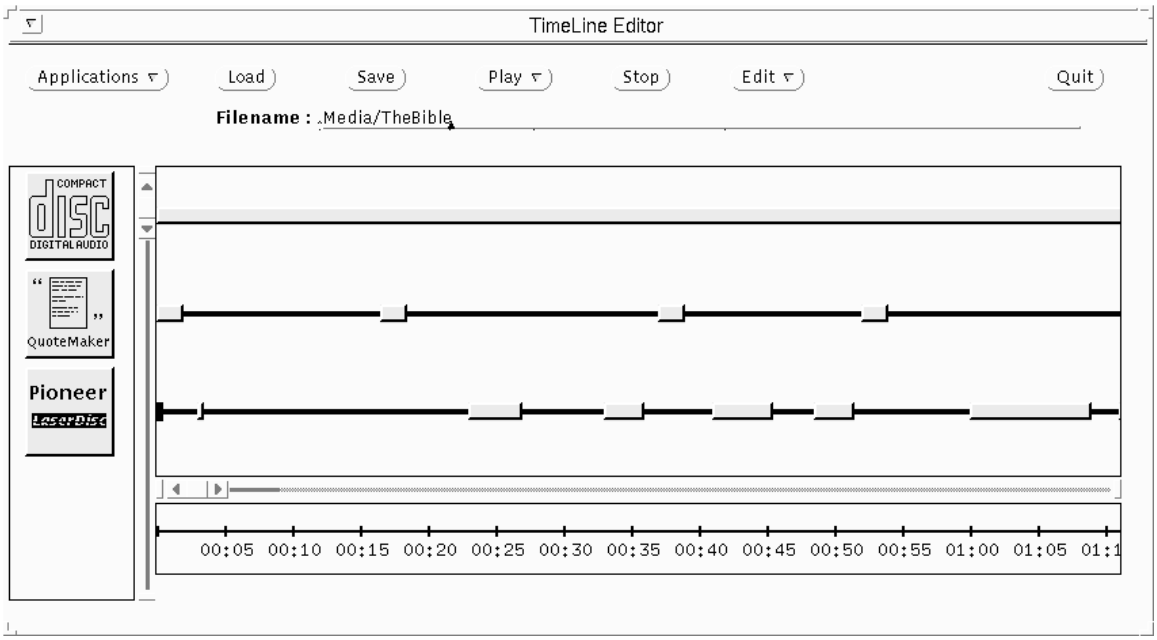

Figure 1: *A TimeLine Editor document coordinating three media editors: cdEdit, QuoteMaker, and videoEdit.*

3.2 The Authoring Application

An authoring application defines the structure of a multimedia document. It provides a paradigm of authorship by offering metaphors and structures with which the author is familiar. The authoring application supports a particular model of authorship just as a media editor supports a particular source of information.

We have chosen to implement a timeline paradigm of authorship for our first authoring application. The TimeLine Editor application represents documents as a number of "tracks" of time, one track for each medium in the document (see Figure 1). It is a simple model that works well for linear presentations requiring no interaction on the viewer's part. Using a timeline model allowed us to concentrate on authorship of documents (as opposed to interacting with already existing documents).

The authoring application does not directly control media; instead, it controls the actions of the media editors which do the actual media manipulation. The authoring application exerts this control via the inter-application messaging system. The interaction between the authoring application and the media editors is explained in Section 4.

3.3 The Inter-Application Messaging System

At the heart of **MAEstro** is an inter-application messaging system similar to the Speaker-Listener protocol used on the NeXT computer. Unlike Speaker-Listener, the **MAEstro** messaging system is currently implemented with Sun RPC's [4], making the **MAEstro** messaging system available on a wide variety of systems.

Each application in the **MAEstro** environment (media editors and the authoring application) uses the inter-application messaging system for communication with other applications. Messages are provided for applications to advertise their services, to request state information from other applications, and to request other applications to open documents and perform media segments within documents. A typical use of the protocol is for an authoring application to request a media editor to open a document, select part of that document, and perform that selection. The messages are few and simple, designed to transmit as little data among applications as possible.

Note that the **MAEstro** protocol does not address the problems of synchronization of media and guaranteed network delivery of continuous media [8, 9]. Our programmers follow specific guidelines in using the protocol to help insure synchronization, but trying to take into account the trade-offs between varying speeds of media hardware and real-time performance requirements is outside the scope of **MAEstro** in its current state.

The semantics of the **MAEstro** protocol are described more fully in Section 5.

3.4 The Port Manager

The Port Manager application is similar to the Sun "portmapper" program in function — it listens on a well-known TCP/IP protocol port for messages from applications that wish to advertise their services, and keeps an internal list of the TCP/IP port numbers passed in by the registering applications. The Port Manager serves as a rendezvous point for applications that wish to communicate with each other. The main difference with the Sun portmapper is that the Port Manager associates TCP/IP port numbers with program names, whereas the Sun portmapper associates TCP/IP port numbers with RPC tuples. Associating port numbers with program names allows multiple applications to use the same RPC protocol.

The Sun portmapper program was written with the traditional client-server model of computing in mind, meaning that only one application is set up to respond to messages sent using a particular RPC protocol (Sun RPC supports message broadcasts where any number of programs may respond, but the assumption is that all responding programs are clones of the same server). The portmapper associates RPC tuples of the form <*program number, version number, procedure number*> with TCP/IP protocol port numbers; the implicit assumption is that only one application will respond to any given RPC. This discourages sharing of RPC's among peer applications that provide similar services.

In the **MAEstro** environment, all applications can provide similar services and therefore speak the same protocol; any application can be both a "client" and a "server". The **MAEstro** Port Manager associates port numbers with program names instead of with RPC tuples, since all applications speak the same protocol. In the Sun client-server model, when a server registers itself with the Sun portmapper, the server tells the portmapper the port number and RPC numbers on which it is listening [6]. In the **MAEstro** model, when an application launches it registers itself with the Port Manager, indicating the name of the application (e.g., the compact disc editor registers itself as "cdEdit"), the hostname on which the application is running, and the port number on which it is listening for incoming messages.

Since all applications in the **MAEstro** use the same set of RPC's, applications can easily communicate with new services (applications) added at any time, since the new services use a protocol already known to the other applications. Applications can become aware of new services by querying the Port Manager.

4 Model of Authorship

To create a multimedia document, an author uses one application for each medium to be included in the final document plus the authoring application used to structure the media. The data itself stays in the media editors, which send representations of their data to the authoring application. In other words, the authoring application does not directly store media but instead stores pointers to media. The Sun Link Service [5] follows this paradigm of linking documents.

Central to the **MAEstro** model of authorship is the notion of a *media segment*. A media segment is part of a document managed by a media editor. For a text editor, a media segment would be a selection of text that might be anything from a single character to a chapter in a book or more; for a videodisc application a media segment would be expressed in terms of frames of video.

Consider the following scenario as an illustration of the model of authorship we envision for **MAEstro**: A music student is to analyze a Beethoven symphony as a term project. The student has collected material for the project from several sources — a performance of the symphony available on compact disc, a videodisc with a performance of the work by a major symphony orchestra, and historical material from several textbooks.

The student inserts the CD into the workstation's CD player and opens the cdEdit media editor. The student uses cdEdit to listen to the performance, stopping along the way to mark passages that she might wish to use as musical "quotes" in the final paper. The student may listen to the disc several times, adding passages to the edit list. It is important to note that at this point the student is collecting notes to be included in a rough draft, so the quality of the edits need not be perfect; rough starting and ending points will do. When finished, the student saves the cdEdit document and moves on to the video material.

The student now launches videoEdit to watch the videodisc performance of the work. The student uses videoEdit in the same way she used cdEdit; she watches the performance, stopping occasionally to add

interesting segments to an edit list. The student finishes making a rough edit list and saves the videoEdit document, then launches the word processor to write the textual portion of the paper. When a rough draft is finished, the student saves the document.

At this point in the authoring session, the component media are roughly organized but have not yet been combined into any overall structure. The student now launches the TimeLine Editor to provide that structure.

The student selects an edit from cdEdit, then clicks on the appropriate track in the TimeLine Editor. A bar appears representing the cdEdit segment's duration. The student places media segments on different places in the timeline, selecting a media segment from one of the editors then clicking on the place in the timeline where the segment is to be performed. When a timeline document has been laid out, the student presses the "Play" button to observe the presentation. The TimeLine Editor sends messages to the media editors at the appropriate times, telling them to perform their segments.

The student observes the timeline she just created and finds that a particular segment of video was a little too long. She goes back to videoEdit and plays the segment, then changes the start and end points until she is satisfied with the result. She re-saves the videoEdit document, goes back to the TimeLine Editor, and replays the document to observe the effect of the modified video segment.

The process of refinement continues in this fashion — the student uses the TimeLine Editor to perform the document, then uses the media editors to do fine tuning of individual media segments, then again to the TimeLine Editor to observe the changes until she is satisfied with the presentation.

5 The MAEstro Messaging System

New media may use as yet unknown data formats, making it difficult if not impossible for an authoring system to support every medium's data format. Thus, the **MAEstro** protocol was designed as a "remote control" protocol instead of a data transfer protocol. By "data transfer protocol" we mean a protocol designed for the transmission of digital media from one application to another. By "remote control protocol" we mean a protocol used to send commands from one application to another. Using a data transfer protocol, an application would transfer digital audio data to a remote application by sending the data directly to the remote application. For two applications to transfer digital audio data in the **MAEstro** environment, an application sends a message asking the remote application to open the document containing the audio data instead of sending the audio data directly. The difference in protocols is analogous to copying memory from place to place (the data transfer protocol) versus copying a pointer to the memory (the remote control protocol).

An individual media segment is represented by:

- Application name: The name of the application (media editor) that created the segment.
- Document name: The name of the document containing the segment.
- Segment information: A representation of the segment itself. The **MAEstro** term for this is a *selection*.
- Duration information: An estimate of the time necessary to perform the segment.

Applications in the **MAEstro** environment define their own notions of "document", "selection", and "performance". However, the notion of time taken to perform a selection has a concrete interpretation shared by all **MAEstro** applications, and is measured in milliseconds. The flexibility and power of the **MAEstro** protocol lies in the ability of each application to define its own notions of document, selection, and performance.

Section 5.1 describes the `Selection` data structure used to represent **MAEstro** selections. Section 5.2 discusses the set of messages sent among applications in the environment. Section 5.3 gives some examples showing how different applications might interpret the notions of document, selection, and performance. Section 5.4 shows how a multimedia document can be constructed from media editors on several hosts simultaneously.

5.1 Selections

The authoring application does not concern itself with specific data formats used by each media editor; instead, it asks media editors for representations of their data. These representations are stored in a data structure called a `Selection`, which looks like this:

```
struct Selection
{
  int start;
  int end;
  int duration;
};
```

An application programmer decides how the application will encode media segments into the two fields `start` and `end`. When the authoring application asks a media editor to perform a selection, the media editor must be able to decode the `start` and `end` fields to identify the original media segment.

The `duration` field serves as an estimate of the time needed to perform a selection, and is specified in milliseconds. This is the only field of the `Selection` structure that is interpreted by the authoring application.

5.2 Messages

There are two types of messages used by **MAEstro** applications: messages sent to other applications and messages sent to the Port Manager. Messages sent to other applications are used for opening documents and performing media segments. Messages sent to the Port Manager are used to register services or to ask for information about application services currently being offered.

During authorship of a document, the authoring application asks a media editor for the name of its currently open document and the current selection within that document. **MAEstro** supplies two messages, `GetCurrentDocName` and `GetSelection`, for this purpose. During performance of a document the authoring application asks a media editor to open a particular document, to select part of the document, then to perform the selection. The messages `OpenDocument`, `SetSelection`, and `PerformSelection` are provided for this purpose. Any application may send these messages to any other application, but at present only the authoring application sends these messages.

An application advertises its services by registering itself with the Port Manager. When an application is about to quit or no longer wishes to advertise its services, it must "unregister" itself with the Port Manager. The protocol provides the messages `ConnectWithPortMgr` and `DisconnectFromPortMgr` for these two actions.

The authoring application needs to be aware of applications currently advertising their services. To obtain this information the message `GetOpenApps` is sent to the Port Manager; the Port Manager returns a list of applications currently advertising their services. Any application may send this message to the Port Manager, but typically the authoring application is the only one concerned with which services are currently available. To ask for a specific service by name, an application sends the message `GetPortFromName` to the Port Manager.

5.3 Interpreting Messages

Media editors interpret the notions of document, selection, and performance as best fit their particular media. This section describes how three applications (a text editor, cdEdit, and the Searcher) interpret these three notions.

5.3.1 Text Editor

A document for a text editor is the paper, memo, letter, etc., typed in by the author. When the text editor receives an `OpenDocument` message from another application, the text editor interprets the string passed to it as the name of a file to open. If the file does not exist, the text editor returns an error code.

Selections are represented as the starting and ending bytes of a sequence of text within the current document. When the text editor receives a `SetSelection` message, the text editor tries to highlight the range of bytes specified by the incoming message. If there is no current document or the given range is invalid, the text editor returns an error code.

The notion of the time necessary to perform a selection is less clear. A text editor does not normally associate time with the text in a document. However, since the protocol requires an application to return an estimate of how long the current selection will take to "perform", the text editor has two choices: return a hard-coded value of perhaps zero milliseconds or one second, or use a heuristic to estimate performance time. The heuristic might be: "based on an average reading speed of 250 words per minute and the length of the current selection, calculate the number of seconds to read the current selection." The **MAEstro** text editor reports a hard-coded value to the authoring application and displays the selected text in another window (in large type), but does not remove the display window when the time has elapsed. A future version of the text editor will provide the author a choice between a heuristically-derived duration and the author's own duration.

5.3.2 cdEdit

The cdEdit application stores documents as edit lists. In addition to start and end points, cdEdit stores a brief text label to describe each edit.

The cdEdit application uses only the `start` field of the Selection structure, ignoring the `end` field. The `start` field is used as an index into the edit list. The `duration` field represents the time to play the edit.

Performance for cdEdit is clear; performing a selection means playing from the start of an edit to its end.

5.3.3 Searcher

Searcher documents store the name of the text being searched and a list of queries the author applies to that text. When a Searcher document is opened, the requested text is opened and the author can redo any of the previously saved queries or she can generate new queries.

A selection is a pointer to one of the queries in a document and a pointer to one of the results, or "matches", from that query. The Searcher uses the `start` field of a `Selection` as an index into the list of previously saved queries, and the `end` field as a pointer to one of the results generated by the query. For example, a selection's `start` and `end` values of 3 and 14 would be interpreted as the third query of the current document and the fourteenth match generated by that query.

The Searcher performs a selection by sending a query and displaying the results. To perform the selection indicated above, the search engine would execute the third query of the current document then display the fourteenth result of the query.

5.4 Network Control of Media

Authorship of multimedia documents is not limited to a single host; the authoring application can coordinate media editors on multiple hosts. When an application registers itself with the Port Manager, it uses the following structure, called a `Port`:

```
struct Port
{
  char* hostName;
  char* appName;
  int   portNumber;
};
```

The `hostName` field indicates the name of the host on which the application is running. The `appName` is the name under which the application advertises its services. The `portNumber` is the TCP/IP port number on which the application is listening.

When the authoring application sends the `GetOpenApps` message to the Port Manager, the Port Manager returns a list of `Ports`, one `Port` for each application that is currently listening for messages. An application

can register itself with any Port Manager on the network, and an application can ask any Port Manager which applications are registered with that Port Manager.

Viewing a multimedia document elsewhere on the network is a potential problem since some media cannot be performed over the network. For example, we cannot play CD audio over our networks although text can easily be displayed over the network with the appropriate window system. To address this problem, the authoring application keeps track of the "authorship host", the host on which a document was originally created. If the authorship host is different than the host from which the document is being played, the authoring application allows the author to specify which host(s) should be used for playback of each media editor in the document.

Perhaps the greatest disadvantage of this type of network control of media is that the network is not completely transparent. Authors should not be forced to think about the network, but considerations such as the location of files and media hardware connected to the local workstation affect transparency. While it is fortunate that the author can specify authorship and playback hosts, it is unfortunate that she must sometimes do so. Of course, if we assume a standard hardware and software configuration for delivery, the problem of network opacity diminishes since all services would be available on all hosts. We can do this to some extent, but some media (such as video) are still expensive, so not all of our workstations will have all media capabilities.

6 Future Directions

MAEstro is on a development schedule that will see its first delivery to the Stanford computing community in October. First delivery of **MAEstro** will take place at two sites: our multimedia lab will provide a variety of video and audio media, and our public workstation rooms will provide baseline services accessible to all. The authoring environment in the public areas will not be as rich as for the multimedia lab, but students will still have access to software-only media (text, search engine data, images, etc.).

Section 6.1 describes the applications currently being developed for the environment. Section 6.2 discusses our goals for ease of use and how we intend to obtain those goals. Section 6.3 discusses plans to extend the environment.

6.1 Future Applications

Although we support some form of text, audio, and video, the environment as defined by the applications above are mostly read-only in nature. Our belief is that while CD audio and videodiscs have value, the availability of writable media will afford a much wider community of authors. In keeping with this philosophy, we are developing the following applications:

- **vcrEdit**, an application to control the new NEC PC-VCR videotape recorder that accepts VHS tapes recorded on any consumer VCR.
- **DigitalTapeRecorder**, an application to record and playback sampled audio on Sparcstations.
- **ShellEdit**, an application that allows the author to spawn Unix system processes, shell scripts, etc. This is a bridge between applications that speak the **MAEstro** protocol and those that do not.

In addition to these applications, we have plans to write a NeXT-to-**MAEstro** protocol converter. The protocol converter will speak both the **MAEstro** and Speaker-Listener protocols, allowing NeXT applications to be used as **MAEstro** media editors.

We are also seeking co-developers to add media editors and alternative authoring applications in order to provide access to a wider variety of media and several paradigms of authorship from which potential authors can choose.

6.2 Ease Of Use

We are working in several ways to make **MAEstro** a simple environment to use. First, we will continue to enhance the abilities of our existing media editors, providing richer functionality to authors. Distributing

authorship among applications can work, but only if the components themselves are fully functional. Second, we will develop additional media editors as new media become available and as demand for other media becomes apparent. Third, we are trying to leverage applications with which authors are already familiar in order to reduce the learning curve. This means adding the **MAEstro** messaging system to existing applications when possible.

6.3 Extending the MAEstro Messaging System

Preliminary use of the environment has pointed out a number of messages that should be added to the protocol. Included are messages to control performance as it is happening (e.g., pausing and resuming a performance, and performing part of a selection), and messages to help overcome "window overpopulation", a cluttering of the screen that increases as the number of open applications increases. Such messages might include requests for an application to hide itself from the screen, to show itself, and to bring itself to the front of the window stack.

The current selection paradigm does not allow for overlapping selections; one selection for a particular medium must end before the next begins. One solution we are considering is additional dialog between a media editor and the authoring application about whether the media editor has the ability to start new selections while it is currently playing a selection.

The NeXT Speaker-Listener protocol is written in Objective-C and allows programmers to add messages by subclassing the two messaging objects. **MAEstro** does not yet have this flexibility, since its messaging system is built on top of Sun RPC which is written in C and is not subclassable. We hope to find a subclassable messaging mechanism to replace the current Sun RPC model. This would allow application programmers to add their own messages to be shared among a select group of applications, and would allow us to experiment with some of the additional protocol items discussed here without breaking the current environment.

7 Conclusions

MAEstro is still in the development stage; we plan to deliver the first version of the environment in October 1991. During the summer we will continue testing with a small number of authors and cycle feedback into the development process.

Section 7.1 discusses some of the benefits of the **MAEstro** model not already mentioned. Section 7.2 discusses some of the problems with **MAEstro**.

7.1 Benefits

The distributed nature of the environment allows third party vendors to enrich the authoring environment by providing full-featured applications. The environment is only as good as its media editors; therefore, the better that applications behave, the better the quality of resulting multimedia documents.

The **MAEstro** model encourages authors to use familiar tools when creating documents. By contrast, centralized authoring environments require the author to use the tools provided by the authoring application to manipulate media. For example, to edit text in Hypercard an author must use the text editing provided by Hypercard (plain text can be imported from the system clipboard, but formatted text from Microsoft Word cannot be imported). This increases the learning curve for the author, since she must learn a new text editor for each authoring application she uses. In **MAEstro**, the author can use the word processor directly in the multimedia document.

MAEstro allows for rapid prototyping of multimedia systems. Multimedia support on workstations is still largely in an experimental stage, making it difficult to predict which media types will prevail in the next few years. The ability to plug in new media without disrupting existing functionality allows **MAEstro** to track the moving target that is the workstation multimedia market.

7.2 Problems

MAEstro does not address well the problem of synchronizing media. There are no synchronization primitives in the protocol, and the environment is currently at the mercy of slow media such as videotape. Macromind's

MediaMaker addresses synchronization by supporting only media that can be interrupted at any time; a text search engine or database tool would not be allowed in the MediaMaker environment because a query cannot be interrupted in the same way that digital audio can be stopped. The result is that real-time presentation is not at all guaranteed; in fact, it is not even a primary goal of MediaMaker. We are trying to ameliorate synchronization problems by enforcing a set of guidelines when developing applications and by adding a preview feature to the TimeLine Editor that would compare a performance's actual behavior to its expected behavior and try to correct for the differences.

The environment does not support version control of documents, as does the Sun Link Service and Apple's Publish-Subscribe model of inter-application communication [7]. There is no way to tell the authoring application to update its link to a media editor's document; if the media editor's document has been changed or deleted, the author must make a new link from the authoring application herself.

Multimedia documents in **MAEstro** tend to have too many components, thus complicating delivery. Recalling the authoring scenario from Section 4, there were at least four documents involved (text, cdEdit, videoEdit, and authoring application document). Delivery of such a document would be problematic in any system since "hard" media are involved (a CD and videodisc would have to be delivered with the documents), but packaging a number of files for delivery has different implications than sending a single file regardless of whether hard media are involved. Part of this problem is handled by network control of media, but there are some media that are not yet amenable to network access; the media themselves must be at the viewer's desk, and it is these media that create the delivery problem. Perhaps the **MAEstro** model joined with applications capable of delivering continuous digital media will alleviate the delivery problem.

The authoring environment in its current state suffers from window overpopulation. Possible solutions are to increase screen real estate (but to what extent? 6000 by 4000 pixels? A wall-sized display?), or add window manager-like messages to the protocol that allow the authoring application to tell applications to hide themselves when not being used, to show themselves when needed, and to hide unnecessary windows during performance of a document.

We have not yet addressed security issues raised by the ability to use media editors over the network. We do not see security as a serious problem in the short term (the X Window System as delivered by the X Consortium is an example of an environment that has so far survived with a limited security scheme), but this by no means diminishes the serious nature of the problem.

8 Acknowledgments

Many thanks go to Teck-Joo Chua for the excellent work he has done implementing the TimeLine Editor and the first versions of most of the other applications in the current environment, as well as his sound advice on current development issues. We would also like to thank Bryan Clair for his help in the early design and implementation phases of the project. Thanks go to the development team of Derek Lai, Wee-Lee Lim, Bryant Marks, and Mark Warren for their work. Additional thanks to David Finkelstein, Yaron Gold, John Interrante, and Johnson Lam for their valuable advice. Special thanks go to Steve Loving for his ideas of student authorship that inspired the project.

References

[1] *NeXT System Reference Manual — Release 1.0 Edition*, NeXT Computer, Redwood City, CA.

[2] Kevin Harney, Mike Keith, Gary Lavelle, Lawrence D. Ryan, Daniel J. Stark, "The i750 Video Processor: A Total Multimedia Solution," *Communications of the ACM*, 64-78, April, 1991.

[3] G. David Ripley, "DVI — A Digital Multimedia Technology," *Communications of the ACM*, 811-822, July, 1989.

[4] *Network Programming Guide*, Sun Microsystems, Inc., Mountain View, CA.

[5] Amy Pearl, "Sun's Link Service: A Protocol for Open Linking," *Proceedings of ACM Hypertext '89*, November, 1989.

[6] W. Richard Stevens, *Unix Network Programming*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[7] J. Paul Grayson, Chris Espinosa, Martin Dunsmuir, Mike Edwards, Bud Tribble, "Operating Systems and Graphic User Interfaces," *SIGGRAPH '89 Panel Proceedings*, December 1989.

[8] Domenico Ferrari, "Guaranteeing Performance for Real-Time Communication in Wide-Area Networks," U.C. Berkeley Technical Report UCB/CSD 88/485, December, 1988.

[9] Domenico Ferrari, "Real-Time Communication in Packet-Switching Wide-Area Networks," International Computer Science Institute Technical Report TR-89-022, May, 1989.