MediaWeaver – A Distributed Media Authoring System for Networked Scholarly Workspaces

Sha Xin Wei*
Stanford University

June 2, 1997

Abstract

We describe MediaWeaver – a software framework for composing distributed media in the context of university research and instruction. Authors compose networked media, software tools and mediastreams, and can freely annotate media by media of any form using schema of their own design. Faculty and student authors compose distributed media using common Macintosh, World Wide Web and NeXTSTEP applications, supported by services from UNIX workstations.

The MediaWeaver system mediates between network multimedia services and interface kits with which novice programmers and non-programmers may easily create radically different interactive views into shared mediabases. The network services include search engine abstractions, filters, relational modeling frameworks.

MediaWeaver has supported collaborative projects in history, drama, music, art, anthropology, environmental studies, and other fields since 1993. Applications range from traditional relational text databases and indexed HTML WWW sites to course readers, research archives, journals and seminar spaces.

*Sweet Hall 415, Stanford University, Stanford, CA 94305. *xinwei@stanford.edu*

- 1. Introduction
- 2. The Problems
- 3. Design Principles
- 4. Architecture
- 5. Future Work
- 6. Bibliography

1 Introduction

A major challenge facing designers of networked computing environments today is to fashion scholarly workspaces which are simultaneously coherent, easily reconfigurable, efficiently expressive – small gestures go a long way, and above all, worth using.

In this paper, we describe MediaWeaver, a system that has streamlined the composition of arbitrary renderable media, mediastreams and applications in diverse models and narrative structures. MediaWeaver is designed to support the construction of models of human systems which are both conceptually rich and data rich. It also mediates between coherent, customizable interfaces and an open set of network services, such as database engines, WWW servers, fulltext and image search engines (Wang [13])[1], and media conversion facilities. And it is designed for open sets of media that will change over time.

Our context is humanities computing (Thaller [12]), which significantly stretches the envelope of networking technology, multimedia, intelligent search systems, and human-computer interface design. Software technology paradigms now run the gamut from verb-object tools ("set the color of the selected word to red") to document processing, intersubjective computing and urban design. (Alexander [1]) We take a perspective situated somewhere between intersubjective computing and urban design. Our method has been to have designers/programmers work intimately with the faculty and student researcher/authors who use the evolving systems.(Ehn [4]) In fact, MediaWeaver was conceived in the beginning as a framework to accelerate our

[1]Note added in proof, we have integrated a suite of image search engines incorporating new work based on wavelet methods.

own multimedia designers' work in creating rich complexes of media supported by relational data models. But it was natural to extend the role of the designer to include authors who were experts in fields outside computer engineering.

2 The Problems

After about five years of making interactive multimedia applications, we took stock of our work process to see where the bottlenecks were, and also what were the greatest defects in the interactive titles produced for scholarly applications.

- Media were scattered all over the network. It was becoming hard to keep inventory using ad hoc databases.
- Researchers significantly changed their conceptual models over the course of a project, so that custom data structures had to be re-written.
- User interfaces had to be constantly re-designed in concert with graphics artists, programmers and researchers, using unpredictably varied media. New interface constructs such as help sprites and custom gestures which did not fit pre-fabricated window-menu-button widgets had to constantly invented.
- Finished titles were often locked into a videodisc or piece of software (eg. Director or Supercard stack), and put out of reach of re-purposors.
- Finished titles had thin media content/ hard content boundaries – users quickly hit the boundaries of what was recorded on a CD ROM or videodisc.
- Conceptual models were often too simplistic to be taken seriously by any but the most novice students. We wanted environments which could support research level work as well as introductory classes. (In general, software which was designed specifically for a given class or lesson was often too rigid or shallow.)

- Hypertext/media graph topologies were either navigable but too sparse to sustain a viewer's interest, or rich but too dense to be comprehended. Hypertext links are fragile, difficult to author or manage, and hard to map.

- We could not easily support multi-author and multi-player discourse networks.

The MediaWeaver was designed to address all of these problems. Its various frameworks were designed to be used by faculty and student authors and by designers of multimedia simulations. It was designed explicitly to be usable by members of academic disciplines outside computer science and engineering. And it had to leverage tiny application programming resources.

We started with two prototype projects in 1993-1994: a history of Renaissance (Elizabethan) theater, and a study of high technology in the Silicon Valley. The first was chosen from a pool of faculty projects which required some management of art images and associate music or text on the network, The second presented the challenge of dealing with a significant, changing body of structured text in a complex, evolving research project. In addition, we wanted to lay the foundation for general relational modeling of human systems as such data became available in the course of the research. In both cases, we could not assume a fixed interface or conceptual model. Indeed, the only surety was change.

This genealogy strongly influenced the design principles which we will outline in the following section.

Since then we have continued with the SiliconBase (Lenoir [9]), as the Silicon Valley History project is called, and have added several other communities and mediabases, including, for example: a prototype for an archive of electro-acoustic music; a Chicana/o artists database (Yarbro-Bejarano [15]); a clearinghouse of international conservation information (Irvine [7]); a history of education since Greco-Roman times (Bloomer [2]); a structural engineering database.[2]

[2]See http://www-asd.stanford.edu/Media1/ASD/website/ProjectsFrame.html.

3 Design Principles and Corollaries

3.1 Make it immediately useful.

Bread and butter reasons, but also participatory design principles suggested that we should let composers start working right away with their own media, conduct seminars and write articles using our system instead of waiting for the Holy Grail. To enable significant scholarly work, whatever we built had to exchange data transparently with commercial applications and databases, and inter-operate transparently with distributed services. Authors were encouraged to use whatever commercial editors they already had on their personal computers (Macintosh or Windows) [3] Our frameworks synthesize commercial, public and custom software. Our authors work in a heterogeneous network where UNIX and Macintosh clients see a common filesystem, and can apply user tools from Macintosh, UNIX (Sun, SGI, NeXTSTEP) to shared mediabases.

3.2 Factor, factor, factor.

The architecture reflects a separation between (1) *persistent storage* in the filesystem (eg. ASCII or AIFF blob bytes) and in databases (eg. blob metadata in Sybase tables); (2) *model* (eg. hypermedia topological structure, bibliography); and (3) *presentation/interaction* (eg. WWW/Mosaic document, Hypercard simulation, custom disposable apps). By decoupling models from media, we can sidestep the question of data ownership and allow complex research models to be constructed on existing corpora or proxy media[4]

Since MediaWeaver stores topological information in databases, it can generate HTML documents dynamically rather than keep source media in HTML files – a simple version of dynamic documents. Factorization gives us the option of interposing even more expressive and nuanced means of forming constellations media or mediastreams on-the-fly.

[3] Typical tools include Microsoft Word, WordPerfect, Adobe Photoshop, Adobe Premiere, Omnipage, DeskScan, Autocad and Mathematica.

[4] We have in mind notions such as using relational grammars to define meta-layouts for user-interfaces. Examples include WRI's Mathematica 2.3, and work by Weitzman and Wittenburg [14]

3.3 Maintain user interface metaphor neutrality.

We wish to allow multiple views on shared media, which means that rather than building a single interface application or layout protocol (*a la* HTML forms), we provide an API supporting multiple, concurrent, and most importantly, reconfigurable interfaces. The MediaWeaver does *not* assume that views must look like word-processors. Word-processor-like document viewers like MS Word or Mosaic present essentially a unidimensional rebus, a stream of generalized characters, some of which are ordinary letters, some of which are raisins of media like an embedded graphic. In contrast, a simulation generally can have quite a different structure, such as a map, timeline, multi-track score, vivarium, video-based telepresence, soundspace etc. MediaWeaver user interface kits do not assume documents, windows, chunks, or links. But the MediaWeaver does deliver documents as a special case. For example, ordinary word-processor documents may be catalogued in indigenous formats.

3.4 Broadcast rather than publish.

MediaWeaver is designed to deliver information over networks, rather than in detached forms such as CD ROM. The CD ROM (and videodiscs etc.) distribution model is in a sense a natural relic of the traditional publishing model which requires a physical commodity in order to function. From the point of view of a university library, most if not all of the same problems encountered in acquiring preserving, cataloguing and circulating paper books or journals recur in dealing with CD ROMs and videodiscs. Some of these library issues are even thornier in the new formats.

Finegrained network distribution of software, even of single computing objects, offers quite a different paradigm which may be more akin to a broadcast model than to the publishing model. This also gives us the flexibilty we need to support live research projects in which the primary source media as well as the secondary literature and even the conceptual models are in flux. In any case, MediaWeaver's factorization allows us to build templates to which we can download a subset of a project's model + data to client. In this way, we can print a standalone version of simulations similar to T. Gieryn's Cornell Biotechnology Lab or G. Crane's Perseus modules ([5]) by downloading data and models from the network into local templates.

Even more interesting are the new modes of dissemination now made possible by online mediabases. MediaWeaver provides a scheme in which progressively more formal or public compositions can arise organically from flexible, personal or project-specific research collections. For example, collections of source material can be acquired and edited according to research agenda. This demand-driven model efficiently allocates human and system attention. New scholarly articles or pedagogical presentations can be made *in situ* and catalogued back into the mediabase. For example, the SiliconBase seminar's reader is an entirely online hypermedia structure which can be modified at any moment by the instructors. Lectures can be composed, presented in conferences, and revised online. Over time, well-critiqued articles can then be given more public status by relaxing their access locks. Such research reports become an online professional journal with the addition of a suitable editorial board and digital signatures. Design issues such as the social conventions around periodicity and cost recovery mechanisms would be interesting to investigate using such a framework.

3.5 Maintain model neutrality.

To allow multiple conceptualizations requires that authors be able to build rapidly several models over the same media. This derives from a practical need to reconcile the very different time-scales involved in designing provisional research schema of annotations and associations vs. designing a MARC-quality archival description of the same set of media. Again, by factorization and abstraction MediaWeaver allows very different communities to work with media, represented when necessary by proxies, using their own models. Consequently, instead of binding to one particular database, MediaWeaver uses a data access framework which allows authors to connect to any of several standard types of RDBM engines over the net, including Sybase and Oracle. MediaWeaver provides an object-oriented abstraction so that its clients need not deal with dialects of RDBMs. Clients can store arbitrary objects like bitmaps or serialized Objective-C objects as meta-data via MediaWeaver's object-oriented database access framework. In practice, (large) media are kept as source media in ordinary distributed filesystems, and (small) meta-data – annotations, references, links, abstracts, etc. – are kept in RDBMs.

3.6 Expect evolution.

Perhaps the key to making an scholarly workspace worth using is to ensure that intellectual content survives across change in technology. This is partly an institutional commitment as well as a technological issue. Aside from the obvious requirement of a modularized architecture whose components may be replaced without breaking service, the following principles guided our work:

3.7 Assume no single data representation.

We do not need to spend resources to converting media systematically to a single format like HTML or SGML. This is perhaps the most important technical feature of MediaWeaver. By making no assumption about the internal structure of a media entity (a blob), and not even requiring that a media entity exists as bytes in a filesystem, MediaWeaver allows authors to compose with any computable or renderable medium whatsoever. This way, MediaWeaver can accomodate currently unknown data types and interactions. Moreover, this way MediaWeaver can deal with opaque or pre-recorded media (eg. TIFF, MPEG, AIFF, TeX, Renderman), performable scripts (eg. NeXTSTEP scorefiles, Mathematica notebooks, Applescripts), executables (eg. a UNIX tool, Hypercard stack, or Java application), and data streams (eg. live video channel) with equal ease /difficulty.

How is this feasible? The working principle here is to –

3.8 Focus on space of transforms more than the base space.

Converting all the authors source media into some standard structure (such as SGML) is not cost effective nor strategic in our context because of the diversity of the material (some conversions would lose too much information), the large human cost (editorial, programmer, administrative), and the constantly changing substance. Moreover, we are not convinced that a universal, permanent (on the scale of decades) document structure exists which can support all the media that authors will use. Therefore, we have decided that it is wiser to build a filter service that can be invoked by MediaWeaver servers as well as their clients.

3.9 Assume nothing about the internal structure of a media entity.

A media entity may be a programmatically generated stream of data, a file of any renderable data type, an executable, or may even exist only as a virtual object in a meta-data record. This allows authors to work with proxy objects even when, for legal or technical reasons, primary media are not available. Conversely, multiple versions of a logical media entity can be tracked. The front end, not the MediaWeaver core, decides how to interpret multiple versions of a blob. For example, a movie clip may exist in MPEG as well as an Apple QuickTime proprietary format. The front end asks for the locally renderable version, but authors deal only with a single logical entity.

4 Architecture

4.1 Media Model

In our model (Figure 1) a logical media entity has a unique tag, zero or more source versions, and usually at least one metadata attribute or proxy. Typically, the media entity is associated to some data in persistent storage, but this is not required. By allowing entities that refer to no source media, we can construct compound media structures. Links between media are stored in a link database. Project designers define their own metadata schema, and may extend the schema as their conceptual models evolve.

Each application project gets its own link and metadata databases so there may be multiple representations overlaying a given set of media. In practice media entities refer to data files (of all types), compiled applications, and URL's.

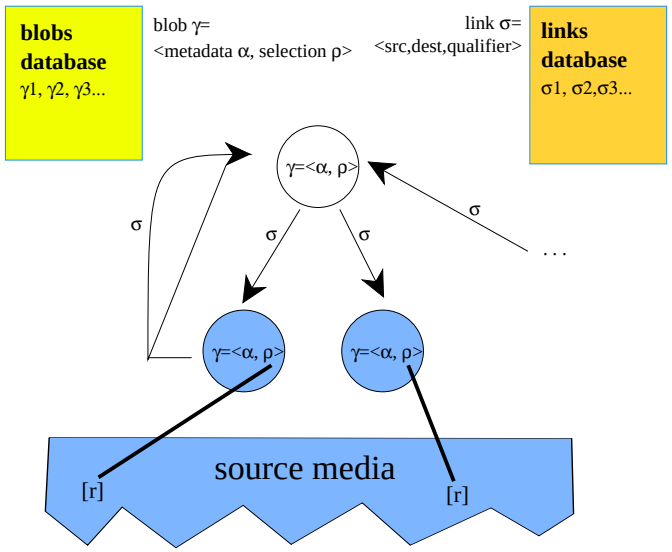

Figure 1. Media Model.

4.2 Framework

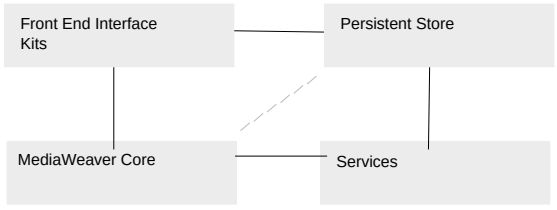

Figure 2. General Architecture – Four layers.

MediaWeaver has four framework layers (Figure 2) which communicate over network protocols:

- a set of user interface kits (Macintosh, World Wide Web, NeXTSTEP/OpenStep),
- a set of mediation classes that abstract over categories of services,
- a set of services (eg. annotation and linking, fulltext search, image search, format conversion), and
- persistent storage (eg. AFS, AppleShare, Sybase, Oracle).

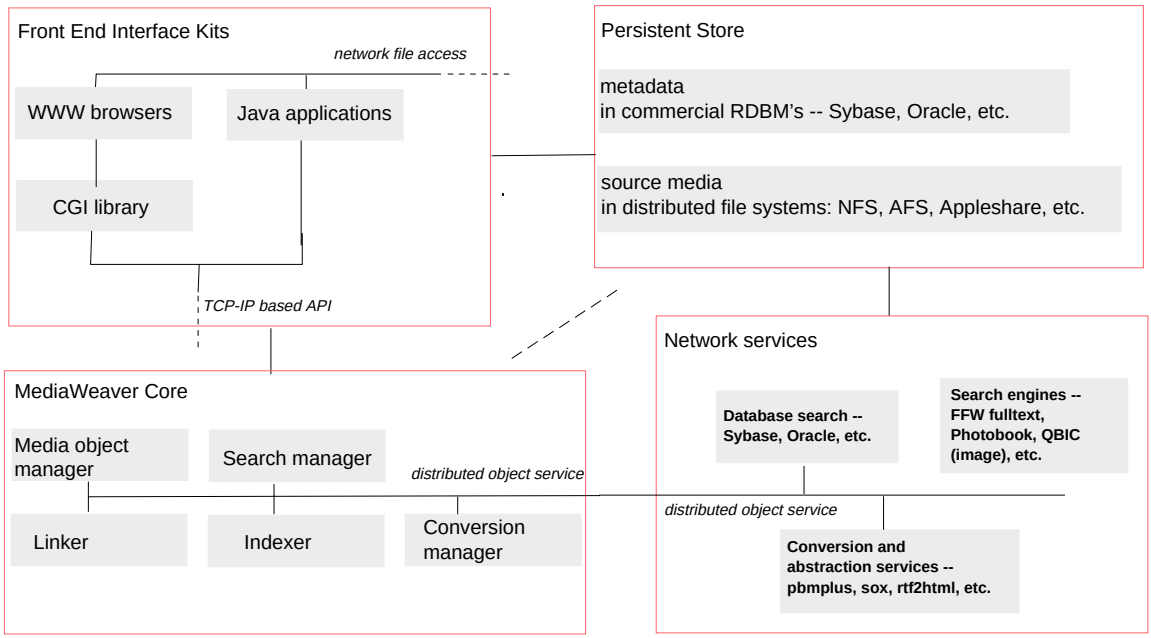

Figure 3. Architecture in detail.

The MediaWeaver's core mediating layer is written as a set of Objective-C classes: Butler, MOM (Multimedia Object Manager), Searcher, Indexer, Linker, Transformer.

- Butler – authenticator/accesser which maps users to equivalence classes of authorship ranging from full creation/destruction of database schema to passive browsing, connects to requested databases and manages MOM, Searcher and Linker objects.[5]

- MOM – multimedia object manager which is the Objective-C representation of multimedia entities' metadata, cacheing descriptors of any type: text, numeric, or any serializable data such as a TIFF, Macintosh PICT or a NeXTSTEP Sound object.

- Searcher – parent query class parses search requests to generate queries against patterns which may themselves be objects. Searcher parcels out a compound search across multiple search engines. Searcher subclasses own SearchEngine objects which connect to particular relational database or content-based search engines to support alternatives such as proximity searches based on special metrics.

[5] In a distributed object system, the Butler plays the role of a proxy for the MediaWeaver core.

- Indexer – can defer or background the creation of indexes that must be performed by particular search engines in order to optimize retrieval.

- Linker – the parent class provides a few convenience functions to trace links based on source, destination, or a string qualifier. Since links are first-class entities in the MediaWeaver, applications may apply MOM and Searcher objects directly to a set of links. This provides great flexibility and extensibility to alternative link management strategies.[6]

- Transformer – convertor and abstractor class uniformizes the conversion of file-based data, including images (pbm, xconv, etc.), text (rtf, HTML, TeX etc.), sound (sox) and video. Using such services, Transformer classes create proxies: compact abbreviations or summaries – image thumbnails, sound snippets, text captions, etc. – of media objects. Child classes handle particular media types.

4.3 Storage, transport of source media and meta-data

Source media are archived as ordinary Macintosh and unix typed files on distributed (AFS) and local file systems. Media streams (eg. live video channels, with local hardware support) are handled as any other blob under this uniform scheme. We anticipate that databased file-systems will eliminate many of the circumlocutions currently required by this file-based storage, but for data portability, we decided against using a uniform proprietary storage mechanism.[7]

The same concern led us to store metadata (annotations, hyperlink maps) in standard RDBM's tables.[8] Although we recognize that this is unacceptable substitute for dynamically adjoining foreign database servers, we see no alternative until certain protocols like Z39.50 or CORBA are standardized and widely supported by data storage and translation facilities.

[6] This is comparable, but at a finer grain, to Hyper-G's [8] link server which also decouples the hypermedia structure from the source media.

[7] For example, we studied NeXTSTEP's Objective-C serialization, Taligent's persistent object store, and Versant and Illustra OODB's, all of which offered advantages over traditional data storage.

[8] We used Sybase and Oracle server for robustness and portability.

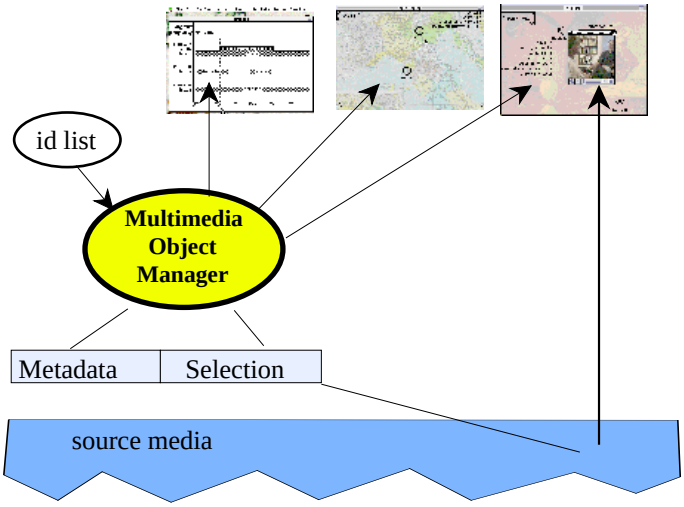

Figure 4. Clients unpack metadata to populate different personal workspaces.[9]

4.4 Interface Kits

Authors can catalog and annotate media in one front end on a particular environment, and have the results immediately accessible via a variety of browsers.

4.4.1 World Wide Web

Our most common front ends are written atop a CGI library that passes requests from the WWW client on to the MediaWeaver server. Results may be assembled into HTML dynamically. The most salient aspect of this is the the sharp decoupling of the connection, query and retrieval logic embodied in the CGI library from the HTML formatting. Rather than building brittle networks of HTML documents, a typical WWW "front end" consists of a few pre-written HTML pages that give context, with forms that send commands to the MediaWeaver via CGI's. The HTML layout templates are stored in text files that are created by designers who know some HTML but are *not programmers*. This factorization has allowed us to deliver MediaWeaver media and services via different WWW browsers' encodings as soon as they appear, and takes maximal advantage of our authors' differing writing skills.

[9] Cf. Figure 7.

4.4.2 Custom Front Ends

Figure 5. Dynamically constructed views of one media entity – Pantaloon – from the Renaissance Project. In the NeXTSTEP application (left), Linked "daughter" images appear as sound and image abstracts on the left, and research assistant's commentary appear below. In the Macintosh application (right), the item of interest occupies the large focus field, with linked media around border.

The particular interfaces shown in Figure 5 contain several experimental navigational methods: tracing links represented by non-text media objects, profile search (query by example), term search in selected meta-data fields, and full content text search. This particularly simple interface emphasizes direct manipulation. A browser can pull a media object into the center of attention, make links, search, and add voice or text annotations. Designers can quickly re-arrange the interfaces without interrupting networked services.

For Macintosh clients, we have built a set of such local data-caching classes and interface classes in HyperTalk, and a set of XCMD's which use the MediaWeaver's API, all packaged into template Hypercard stacks.[10] Under NeXTSTEP we have extended the data access framework's user interface classes in the Interface Builder.(DatabaseKit, EOF [6]) From these components it is quite easy to (re)fashion viewer/composer applications customized

[10]Obviously, a more robust application developer workbench would be desirable, but that will depend on the evolution of authoring environments from innovative industrial partners.

to the needs of our faculty and student authors. We have produced more than a dozen viewer applications tailored to various tasks, such as cataloguing, navigating and weaving compound media, allowing various intensities of reading or authoring. (see videotapes [10], [3])

MediaWeaver front ends support some alternative UI paradigms such as drag service with on-the-fly conversions across the entire network workspace, graphical gathering and winnowing searches, and geographic map operations. They communicate with sister applications via system-dependent data-object exchange services (eg. NeXTSTEP Pasteboard objects or AppleScript). For example selections can be sent to applications such as MacAtlas and Mathematica with graphical or formatted text results returned to originating views.[11]

[11] The NeXTSTEP user interface kit includes a polymorphic MediaView which can display multiple types of media, a CompositeView which performs some layout and tiling of daughter MediaViews, TableViews optimized for matrices of scalar types, as well as a FetchGroup which buffers queries to the MOM and caches returned data.

Figure 6 (below). The same CGI formatters are used to dynamically generate these two different WWW interfaces from end-author layout templates. (Here, viewed in OmniWeb.) In the Chicana Art interface, the themes are themselves are catalogued as *virtual* media entities, and are presented via proxy icons in a "menu" as the result of a query. Resulting pages can be constructed with embedded MediaWeaver commands which are parametrized by arguments extracted from the metadata.

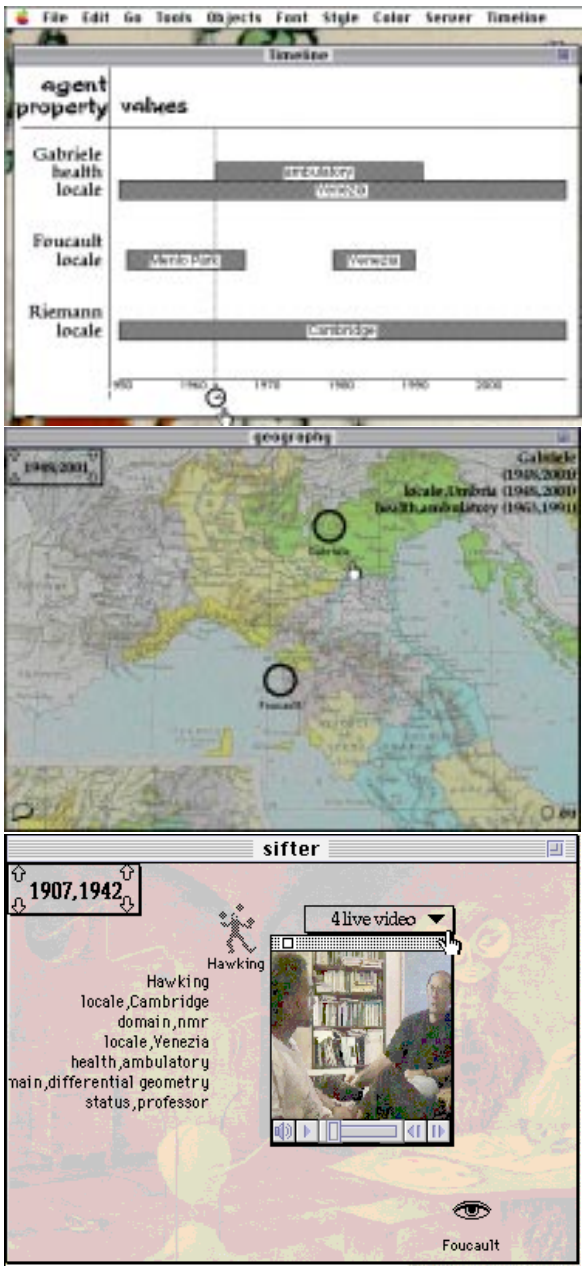

Figure 7. Timeline, map and multimedia bulletin board. – three independent network interfaces (Macintosh Hypercard) to a single model maintained by the MediaWeaver. In this case, the model consists of agents α, attributes σ and

time-periods (t_1, t_2) during which α has property σ. In the last workspace, the author can link media from a live source or prepared file to an agent in the abstract model.

5 Future Work

Now that we have a sufficiently rich substrate of services, and a modest but diverse set of scholarly user communities and corpora, we would like to turn our efforts to make the user environments more seamless. In the near future, we would like connect MediaWeaver front ends with commercial siblings such as GIS apps, and SMPs or numerical engines. We are evaluating multi-architecture, metaphor neutral user interface frameworks which can talk to the MediaWeaver. Apple's OpenDoc and ScriptX application frameworks are possibilities, as is JAVA[12] and various automatic document layout systems.[13]

We will be extending the project in several application areas, including relational models of human systems, and geographically-sited information systems.[14] Project disciplines include art, anthropology, history, literature and theater.

At the lowest level, we are studying implementations of CORBA distributed object protocol to see how best to migrate our rudimentary distributed object service. And we are preparing for a project to feed rich media alongside video service.

6 Acknowledgments

Deborah Zimmerman wrote the Hypercard front ends for the Macintosh. Rick Wong wrote the TCP-IP libraries and implemented the API. Siew Sim wrote the TCP-IP based Object Server. Ying Liang has been responsible for maintaining the third generation core classes as well as the CGI's. Kan Yu-Tung contributed to the CGI interfaces (WWW front ends). And James Ze Wang contributed the image search engines.

[12] See http://java.sun.com/

[13] See Weitzman, MacNeil and others at MIT [14].

[14] See the Alexandria Project [11].

We thank Tim Lenoir, Henry Lowood, Judy Dolan, Bill Eddelman, Yvonne Yarbro-Bejarano, Nickie Irvine, Darryl Weiner, Paul Edwards, Renate Fruchter, Michael Winnick and the student assistants for participating in the design process, and we are indebted to Charles Kerns for his constant interest. We thank also Bill Verplank, Terry Winograd, Rosanna Carotti and Milon Mackey for their criticism, and Brodie Lockard, Michael Levin and Barbara Maliska for their support.

References

[1] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language*. Oxford University Press, 1977.

[2] Martin Bloomer. Classics – a history of liberal education from the greeks to the renaissance, 1995–1996. http://www-leland.stanford.edu/class/ch114/.

[3] Academic Software Development. Video: MMDD on NeXT and Macintosh, Spring 1994. informal notes.

[4] Pelle Ehn. Towards a philosophical foundation for skill-based participatory design. In P. Adler and T. Winograd, editors, *Usability: Turning Technologies into Tools*, pages 116–132, 1991.

[5] ed. Gregory R. Crane. Perseus project: An evolving digital library on ancient greece and rome. Technical report, Tufts University, 1997. http://www.perseus.tufts.edu/.

[6] NeXT Inc. *NeXTSTEP*. Addison Wesley, 1992.

[7] Dominique
Irvine. Latin america environmental information map project, 1995–1996. http://lummi.stanford.edu/class/anthro161a/imp/IMP.html.

[8] Frank Kappe, Keith Andrews, Joerg Faschingbauer, Mansuet Gaisbauer, Michael Pichler, and Juergen Schipflinger. Hyper-G: A new tool for distrobuted hypermedia. Technical report, Institute for Information Processing and Computer Supported New Media (IICM), Graz University of Technology, Graz Austria, 1994.

[9] Tim Lenoir and Sha Xin Wei. Networked scholarly workspaces for history of high technology, talk at MIT, March 1995. http://lummi.stanford.edu/Media2/pix/www/MIT/slides.

[10] Academic Software Development Silicon Valley History. Siliconbase project video, Spring 1994. produced by Chris Madison, SITN.

[11] Terry Smith, Jim Frew, and Qi Zheng et al. Alexandria digital library, 1995. http://alexandria.sdc.ucsb.edu/.

[12] Manfred Thaller. What is 'source oriented data processing'; what is a 'historical computer science'? In *Historical Informatics: an Essential Tool for Historians?*, 1994.

[13] James Ze Wang, Gio Wiederhold, Oscar Firschein, and Sha Xin Wei. Wavelet-based image indexing techniques with partial sketch retrieval capability. *Advances In Digital Libraries*, May 1997. http://www-db.stanford.edu/ wangz/project/imsearch/ADL97/.

[14] Louis Weitzman and Kent Wittenburg. Automatic presentation of multimedia documents using relational grammars. In *Proceedings of ACM Multimedia 1994*, 1994.

[15] Yvonne Yarbro-Bejarano. Chicana art project. *Speaking of Computers*, April 1995. See also Rockefeller grant, or http://www-asd.Stanford.EDU/Media1/ASD/website/blurbs/ChicanaArt.html.