Multivalent Documents:

A New Model for Digital Documents*

Thomas A. Phelps and Robert Wilensky

Division of Computer Science
UC Berkeley
Berkeley, CA 94720-1776

* This work was supported as part of the NSF/NASA/DARPA Digital Library Initiative, under NSF IRI 94-11334.

Multivalent Documents:

A New Model for Digital Documents

Thomas Phelps and Robert Wilensky

Abstract

"Multivalent documents" is a model of documents that addresses some of the shortcomings one currently encounters when manipulating documents in digital form. In the multivalent document model, a document is composed out of distributed data and program resources, called *layers* and *behaviors*, respectively. The model exposes virtually all aspects of document processing to behaviors, and provides the means to compose these components into a single coherent document. Behaviors allow the model to be highly extensible, including the capability to be extended to work with arbitrary document formats. We have implemented the model in Java, and developed behaviors that support multiple document types (scanned page images, HTML, and ASCII) and a number of different user-interface metaphors (e.g., "lenses" and "Notemarks"). The multivalent document model enables one to better use digital documents for tasks in which paper documents are still otherwise superior to digital documents, such as annotating someone else's document. We have shown how the model is naturally conducive to realizing powerful forms of distributed, open annotation by implementing a variety of annotation types, some familiar and some novel.

1 Introduction

Digital documents are superior to their paper counterparts in many ways. They are easier to edit, reproduce, distribute, and search than are paper documents. With the use of networked information systsems, digital documents are also more highly available and can be found more easily than documents in paper format. In addition, electronic documents can manifest properties that have no ready paper counterpart. Commonly available examples of such properties are hyperlinks, virtual structures (e.g., documents whose elements are created dynamically), and inclusion of "dynamic media" (either media types with a temporal extent, such as sound or video, or with an interactive component).

Despite these valuable features, paper is still superior to the digital medium for many purposes. Among the current advantages of paper is its ability to support a large variety of creative manipulations by which the otherwise passive reader becomes actively engaged in a document. We will use the term "annotation" to refer loosely to such manipulations. For non-recreational reading, active engagement with the materials is a key part of understanding. Levy and Marshall eloquently state the case, as observed in their ethnographic study of information analysts [LM95]:

> Annotation is a key means by which analysts record their interpretations of a particular document; in fact, annotation often acts as the mediating process between reading and writing. Analysts generally do not take notes by writing their observations down on a separate sheet of paper or in a text editor .... Instead, they mark on the documents themselves. ... Post-Its ... highlight segments of text ... marginalia ... automatically marked text .... These marking practices increase the value of the documents to the analysts and form the basis for their personal and shared files. ... [P]aper is a valuable medium for recording many types of annotations not readily recorded in a digital medium.

The utility of paper for annotation is reinforced by Sellen and Harper's field study [SH97], in which they found that paper was used almost exclusively as the medium for reading, for reviewing other people's papers and data, and, especially, for tasks involving collaboration, despite a heavy reliance on electronic tools for authoring one's own text.

While there are many reasons for this practice, it is partially due to shortcomings of digital document technology. One can annotate virtually any paper document in arbitrary ways, and share the results with others, all with no appreciable overhead. The same situation does not hold for digital documents. Until it does, the digital medium will be at a disadvantage to paper for certain activities, especially those involving collaboration.

Of course, once the capability to annotate is readily available in digital form, annotated documents would enjoy the many benefits unadorned documents already enjoy from digitalization. In addition, the digital format would provide the possibility of entirely new forms of annotation, since it would allow the exploitation of the dynamic and interactive capabilities of the digital medium. Indeed, entirely new notions of documents might be possible, as taking further advantage of the networked, digital world could enable electronic documents to become something more than digital versions of paper documents.

In this paper, we propose features that digital document models need to support in order to impart to digital document affordances more like those of paper. We use annotation as a paradigmatic case, arguing that models that properly support annotation must support some very general and powerful capabilities. In effect, it is not unreasonable to think of annotation as a "complete" digital document problem, in that models that properly support it must support a number of general and important document properties. We then describe a new model of documents, called "multivalent documents", that provides these general capabilities. We demonstrate the generality of this model by first describing its application to a quite different task, namely, that of enlivening legacy documents. Then we show how the model effects the annotation capabilities we require, demonstrating these capabilities over a number of different document formats.

2 Requirements of Annotations

Here we list a set of requirements for digital annotations. Then we look at the implications of these requirements for digital documents in general. Afterwards, we present and describe a model that we believe largely addresses these needs.

We suggest that an adequate document model to support digital annotations requires the following features:

Appearance *in situ*, yet separate. Annotations should appear on the documents themselves, situated appropriately with respect to their referents. This property contrasts with the use of newsgroups or email messages, in which portions of documents must be excerpted in order to be commented upon. We interpret this requirement as meaning that the annotation should not refer to or be part of a *copy* of a document, which can change independently of the original, but should annotate "the document itself". At the same time, experimental evidence indicates that users want "to regard annotations as a separate layer of the document ... perceptually distinct from the underlying text" [OS97].

High expressiveness. Annotations must have the power to engage with the document deeply, potentially modifying content, appearance or runtime properties. This means that annotations must be available at various grains, from the equivalent of a Post-It™ note to a copy-editor's detailed corrections (*cf.* superficial versus deep annotation [LS93]). In addition, annotations must be able to provide arbitrarily powerful forms of user interaction, providing active capabilities in addition to passive markings. For example, in addition to being able to put passive copy-editing marks on a document, one would like the possibility of executing copy-editor marks to change the underlying document.

Extensibility with composability. Individuals, readers, groups, or third parties should be able to develop their own styles of annotations, which may be highly varied [Mars97]. As new annotation types are devised, it must be possible to seamlessly integrate these with the old. Not only should previous forms of annotations continue to function as new forms are added, but the various forms need to compose together where appropriate.

Distributed and open. Annotation must be an open process, in that one should be able to annotate any document one can view, and share the resulting annotated document with others. While annotations need to appear *in situ* on a given document, it must be possible for individuals to create them without modifying the document per se. It must be possible for the annotator to store the annotation wherever that individual has storage capabilities, and make these available by wherever mechanism that individual makes other resources available. Hence, annotating must require no special access privileges to the target document, or special recognition of the annotator by the target document's server. (We call this mode of interaction "spontaneous collaboration", i.e., the ability to collaborate with small infrastructural overhead.)

Format independence. Users should be able to continue to use their preferred document preparation systems, yet produce documents amenable to annotation. Thus, annotation systems must work with a variety of source document formats, from scanned page images to documents conforming to a markup language.

Platform independence. Since annotated documents need to be heterogeneous distributed networked entities, it is desirable to be able to view the document on a platform other than the one on which either the document or the annotation was created.

Robustness. As an annotation may reside in one place, but refer to a document in another, documents and annotations may change asynchronously. Annotations, therefore, need to be robust enough to survive at least modest document modification.

Some of these capabilities are related to digital document issues well known to the research community. For example, Frank Halasz's [Hal88] seven issues for next-generation hypermedia systems include collaboration support, extensibility and tailorability. Openness and format independence (interoperability generally) are also subject of considerable interest. Nevertheless, while many forms of annotation capabilities now exist, we believe that digital annotation is not a ubiquitous part of computer interaction because each system fails to address one or more of these key requirements. Indeed, most annotation systems are document-type-specific, require modification of the document to include the annotation, and are not readily extensible. That is, they lacking most of the essential properties we stipulate. (We examine such systems in section 9.)

Generalizing from these desiderata for annotations, we can enumerate three basic design principles for a digital document model:

1. **Highly Distributed.** Our first general requirement is that the document model take further advantage of the networked world. That is, the document model should allow documents to be assembled from multiple independent content sources. Having a document comprise multiple composable components is useful for many purposes. The simplest examples of such distributed documents are combinations of style sheets with marked up text, or multiple overlays stored separately. Annotation imposes an especially stringent version of this requirement, as one needs to compose an original document at one source with an annotation by another author at another source, and these must fuse coherently at any grain. A quite different example of distribute sources requiring intimate composition is a document composed from

scanned page images at one source with an analysis of those images from another source. Finally, we require that procedures that extend functionality be available as composable distributed components as well.

2. **Highly Open.** Document components may be distributed, created by different authors using different authoring tools, and kept on different servers with varying privileges. Therefore, the model must accommodate different access protocols, different document formats, and different platforms. Any user should be able to participate with low cost of entry. Together with the capability of composing content from multiple sources, openness imposes the additional requirement that performance be robust and degrade gracefully in the face of changes to document sources, which will happen asynchronously.

3. **Highly Extensible.** Since we want support not for a specific number of annotation types, but for an open-ended number of types, we need a system that is arbitrarily extensible and tailorable, as Halasz suggest. As suggested above, extensible means that features—both content and programs—imported from one source integrate coherently with those imported from a separate source.

3 Multivalent Documents

The multivalent document model (MVD) [PW96a, PW96b] is our attempt to design a document model that realizes the capabilities described above. MVD is an architecture in which a document is viewed as a composition of intimately related but distinct *layers* of content and dynamically loaded program objects, called *behaviors*. Layers and behaviors are assembled by an MVD-compliant browser from multiple distributed sources over the network. MVD provides an infrastructure for the meaningful composition of layers and behaviors.

Potentially any media type can be bridged into the multivalent model. (As illustrated below, as of this writing, MVD operates on scanned page images, ASCII, and HTML.) MVD was designed and implemented by Tom Phelps. It is written in Java, and hence, runs on any Java-compliant platform. MVD behaviors provide extensibility; the multivalent document infrastructure provides a way to integrate such extensions coherently.

3.1 An Illustration

To illustrate MVD, we first describe the application of the model to scanned document images. Scanning paper documents is a standard way to make electronic documents from paper ones. While scanned documents provide a reasonably faithful rendering of the author's intentions, as expressed in the paper format, scanned images are difficult to interact with or manipulate. Because they are so recalcitrant, such a data type is a challenging application for a document model. Here we adopted the goal of enlivening these legacy documents. We will return to our discussion of annotation after this example.

In general, MVD documents comprise layers and behaviors. In this application, each document's layers include (i) scanned document images, and (ii) the result of submitting these images to a document recognition process (optical character recognition, or OCR). In some cases, determined by the content of the page, additional layers may be provided as well. A set of behaviors is included in each document implementing many of the functions familiar in browsers and word processors. Again, some documents contain additional behaviors performing functions that make sense only for that document's specific content.

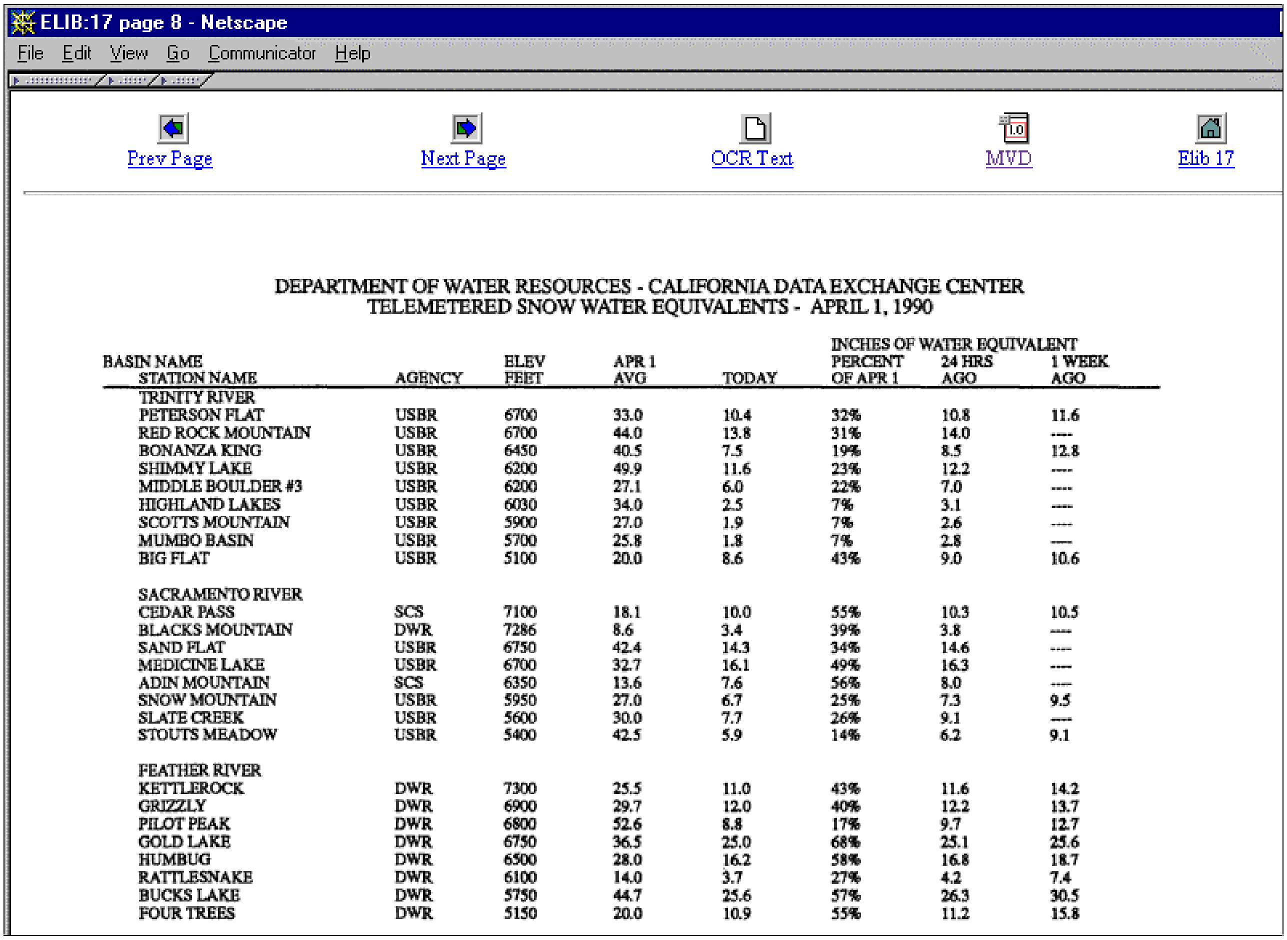
For example, Figure 1 shows in a web browser a scanned page image from the UC Berkeley Digital Library Project server (http://elib.cs.berkeley.edu). Using a standard web browser, one can examine such scanned document images, but no other manipulation or exploitation of them is readily available.

Figure 2 shows the result of clicking the button in Figure 1 labeled "MVD". Doing so provides access to this document as a multivalent document. More precisely, a Java applet implementing MVD is called, and is given as an argument an MVD "hub document". The hub document describes the layers and behaviors that this particular document should contain. In the case of our scanned images, the hub document specifies as layers the images of the document, and the output of an OCR process, which contains both the inferred text of the document and positional information about where these words appear in the image. The behaviors specified in the hub document include a standard set, plus a number of behaviors that are specific to particular documents or pages.

The user is shown the document in a separate window, with a familiar-looking menu bar across the top. The entries that appear in the individual menus will be determined by the particular behaviors loaded. By selecting from the image, or by selecting menu entries, users avail themselves of the functionality provided by the loaded MVD behaviors.

For example, Figure 2 shows some text that has been selected; the background of this text, "CALIFORNIA DATA EXCHANGE" is highlighted in the image to reflect this status. This text was selected simply by a mouse click-and-drag on that portion of the image. Of course, this is exactly the behavior one expects in a word processor, but not from a GIF image, which is in fact that format of the data. As in a text processing system, the text corresponding to the highlight region is placed in the window system's selection buffer, from which it is available for pasting into other applications.

In addition, the user has previously chosen the "Search" entry on the "Edit" pull-down menu. This action called into view the search dialog box, as shown. The text entered into this widget is found and highlighted in the image.



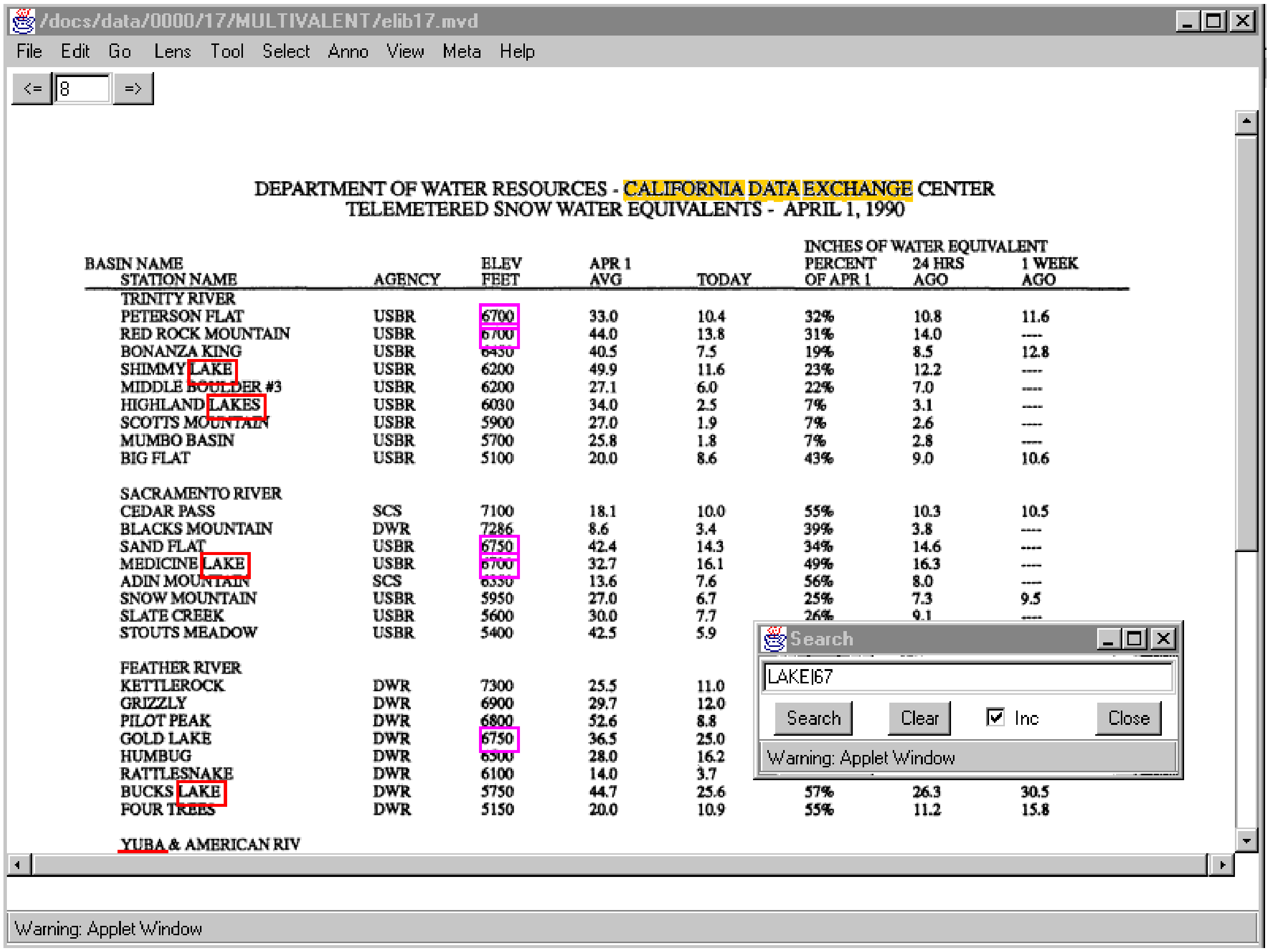
DEPARTMENT OF WATER RESOURCES - CALIFORNIA DATA EXCHANGE CENTER
TELEMETERED SNOW WATER EQUIVALENTS - APRIL 1, 1990

BASIN NAME / STATION NAME	AGENCY	ELEV FEET	APR 1 AVG	TODAY	INCHES OF WATER EQUIVALENT PERCENT OF APR 1	24 HRS AGO	1 WEEK AGO
TRINITY RIVER							
PETERSON FLAT	USBR	6700	33.0	10.4	32%	10.8	11.6
RED ROCK MOUNTAIN	USBR	6700	44.0	13.8	31%	14.0	----
BONANZA KING	USBR	6450	40.5	7.5	19%	8.5	12.8
SHIMMY LAKE	USBR	6200	49.9	11.6	23%	12.2	----
MIDDLE BOULDER #3	USBR	6200	27.1	6.0	22%	7.0	----
HIGHLAND LAKES	USBR	6030	34.0	2.5	7%	3.1	----
SCOTTS MOUNTAIN	USBR	5900	27.0	1.9	7%	2.6	----
MUMBO BASIN	USBR	5700	25.8	1.8	7%	2.8	----
BIG FLAT	USBR	5100	20.0	8.6	43%	9.0	10.6
SACRAMENTO RIVER							
CEDAR PASS	SCS	7100	18.1	10.0	55%	10.3	10.5
BLACKS MOUNTAIN	DWR	7286	8.6	3.4	39%	3.8	----
SAND FLAT	USBR	6750	42.4	14.3	34%	14.6	----
MEDICINE LAKE	USBR	6700	32.7	16.1	49%	16.3	----
ADIN MOUNTAIN	SCS	6350	13.6	7.6	56%	8.0	----
SNOW MOUNTAIN	USBR	5950	27.0	6.7	25%	7.3	9.5
SLATE CREEK	USBR	5600	30.0	7.7	26%	9.1	----
STOUTS MEADOW	USBR	5400	42.5	5.9	14%	6.2	9.1
FEATHER RIVER							
KETTLEROCK	DWR	7300	25.5	11.0	43%	11.6	14.2
GRIZZLY	DWR	6900	29.7	12.0	40%	12.2	13.7
PILOT PEAK	DWR	6800	52.6	8.8	17%	9.7	12.7
GOLD LAKE	DWR	6750	36.5	25.0	68%	25.1	25.6
HUMBUG	DWR	6500	28.0	16.2	58%	16.8	18.7
RATTLESNAKE	DWR	6100	14.0	3.7	27%	4.2	7.4
BUCKS LAKE	DWR	5750	44.7	25.6	57%	26.3	30.5
FOUR TREES	DWR	5150	20.0	10.9	55%	11.2	15.8

Figure 1: A scanned page image inside a web browser

Functionality such as searching and selecting, and most other functionality we have made available for scanned images, will work on any of the (approximately 200,000) pages in our collection. However, some behaviors and layers are sensible only in certain contexts. For example, the page in our example happens to contain tabular data, and we happen to have available both a layer corresponding to a tabular analysis of the image, and a behavior, “Table Sorting’”, that exploits this information. Specifically, one menu entry contributed by this behavior (the “Table Regions” entry under the “Meta” menu) draws a box around the known table region, as well as boxes around active portions of the table header. Clicking on one of these active portions sorts the table image by the data in that column. Figure 3 shows the same page after the table regions have been highlighted, and the table sorted by clicking on the column labeled “TODAY’”.

Functionality like selection, searching and table manipulation is familiar in conjunction with other document formats. However, they are not readily realizable for scanned images. Using MVD, it was reasonably straightforward to provide this functionality, despite the fact that there is nothing in MVD that is specific to scanned images. That is, the exploitation of image and OCR layers needed to achieve the capabilities just illustrated is done by a behavior, not by the MVD infrastructure per se; behaviors like Search are not specific to scanned images, but will work with other data types. We will illustrate the multi-data-type capability of MVD behaviors below.



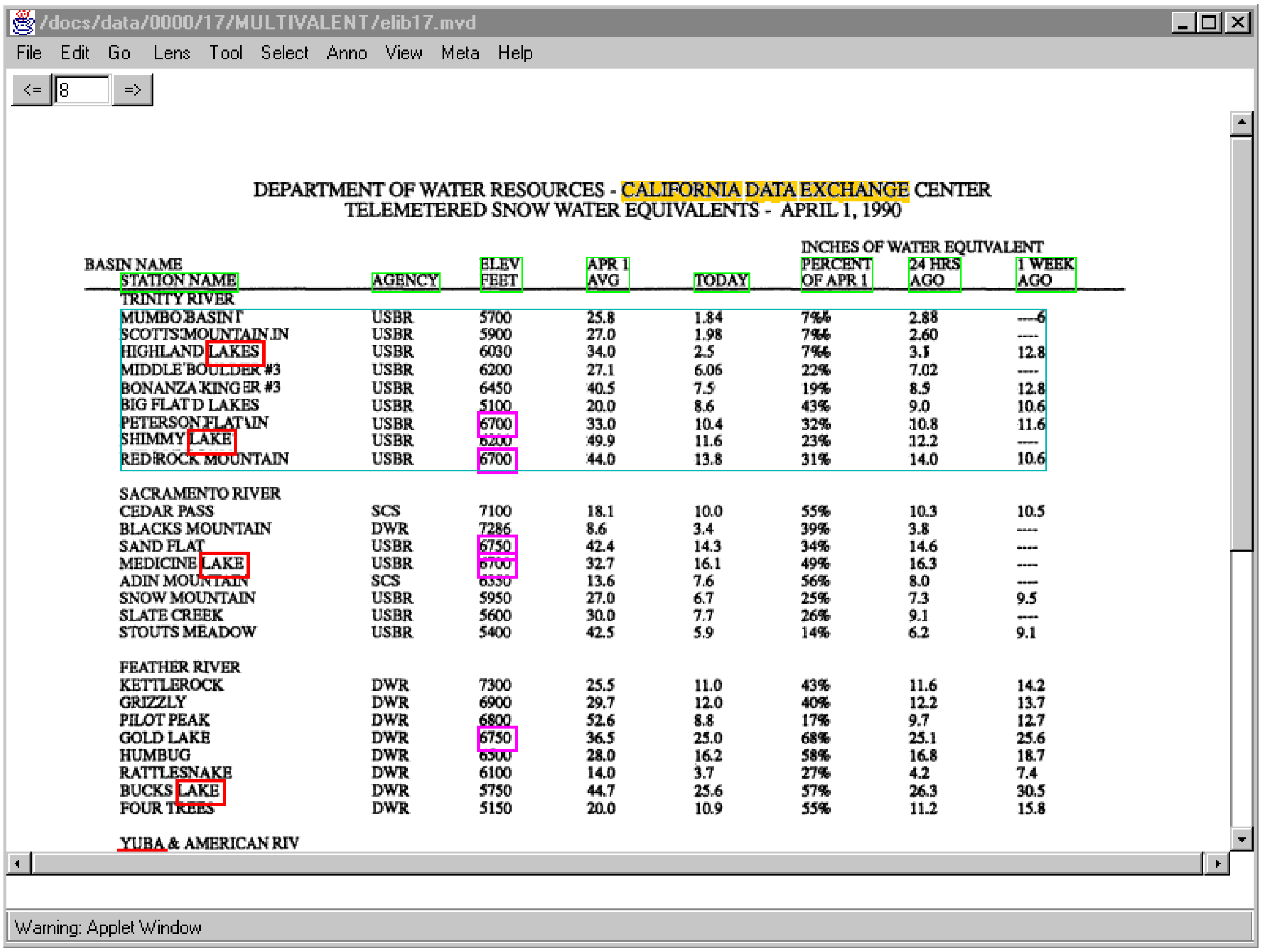
DEPARTMENT OF WATER RESOURCES - CALIFORNIA DATA EXCHANGE CENTER
TELEMETERED SNOW WATER EQUIVALENTS - APRIL 1, 1990

BASIN NAME / STATION NAME	AGENCY	ELEV FEET	APR 1 AVG	TODAY	INCHES OF WATER EQUIVALENT: PERCENT OF APR 1	24 HRS AGO	1 WEEK AGO
TRINITY RIVER							
PETERSON FLAT	USBR	6700	33.0	10.4	32%	10.8	11.6
RED ROCK MOUNTAIN	USBR	6700	44.0	13.8	31%	14.0	----
BONANZA KING	USBR	6450	40.5	7.5	19%	8.5	12.8
SHIMMY LAKE	USBR	6200	49.9	11.6	23%	12.2	----
MIDDLE BOULDER #3	USBR	6200	27.1	6.0	22%	7.0	----
HIGHLAND LAKES	USBR	6030	34.0	2.5	7%	3.1	----
SCOTTS MOUNTAIN	USBR	5900	27.0	1.9	7%	2.6	----
MUMBO BASIN	USBR	5700	25.8	1.8	7%	2.8	----
BIG FLAT	USBR	5100	20.0	8.6	43%	9.0	10.6
SACRAMENTO RIVER							
CEDAR PASS	SCS	7100	18.1	10.0	55%	10.3	10.5
BLACKS MOUNTAIN	DWR	7286	8.6	3.4	39%	3.8	----
SAND FLAT	USBR	6750	42.4	14.3	34%	14.6	----
MEDICINE LAKE	USBR	6700	32.7	16.1	49%	16.3	----
ADIN MOUNTAIN	SCS	6330	13.6	7.6	56%	8.0	----
SNOW MOUNTAIN	USBR	5950	27.0	6.7	25%	7.3	9.5
SLATE CREEK	USBR	5600	30.0	7.7	26%	9.1	----
STOUTS MEADOW	USBR	5400	42.5	5.9	[illegible]	[illegible]	[illegible]
FEATHER RIVER							
KETTLEROCK	DWR	7300	25.5	11.0	[illegible]	[illegible]	[illegible]
GRIZZLY	DWR	6900	29.7	12.0	[illegible]	[illegible]	[illegible]
PILOT PEAK	DWR	6800	52.6	8.8	[illegible]	[illegible]	[illegible]
GOLD LAKE	DWR	6750	36.5	25.0	[illegible]	[illegible]	[illegible]
HUMBUG	DWR	6500	28.0	16.2	[illegible]	[illegible]	[illegible]
RATTLESNAKE	DWR	6100	14.0	3.7	[illegible]	[illegible]	[illegible]
BUCKS LAKE	DWR	5750	44.7	25.6	57%	26.3	30.5
FOUR TREES	DWR	5150	20.0	10.9	55%	11.2	15.8
YUBA & AMERICAN RIV							

Figure 2: A scanned image page "enlivened" as a multivalent document. The text corresponding to "CALIFORNIA DATA EXCHANGE" has been selected. The search widget shows a search for the terms "LAKE" and "67"; words whose corresponding images begin with these strings are highlighted in the image. The highlighting is in color and is much more vivid than in this illustration. (Note that the occurrence of "LAKE" in "GOLD LAKE" has been missed, due to an error in the OCR process.)

The reader may examine the functionality described herein by pointing a Java-compliant Web browser at http://elib.cs.berkeley.edu. The "guided tour" of documents is recommended.

We note in passing that enlivening legacy documents raises the larger issue of the inter-operation of digital and paper documents. Even as digital documents further depart from paper in terms of functionality, the need to inter-operate between the two media types is increasing. The increased functionality of the multivalent document architecture helps address this problem, even as the same enhanced functionality serves to exacerbate it.



DEPARTMENT OF WATER RESOURCES - CALIFORNIA DATA EXCHANGE CENTER
TELEMETERED SNOW WATER EQUIVALENTS - APRIL 1, 1990

BASIN NAME STATION NAME	AGENCY	ELEV FEET	APR 1 AVG	TODAY	INCHES OF WATER EQUIVALENT PERCENT OF APR 1	24 HRS AGO	1 WEEK AGO
TRINITY RIVER							
MUMBO BASIN	USBR	5700	25.8	1.84	7%	2.88	----6
SCOTTS MOUNTAIN	USBR	5900	27.0	1.98	7%	2.60	----
HIGHLAND LAKES	USBR	6030	34.0	2.5	7%	3.1	12.8
MIDDLE BOULDER #3	USBR	6200	27.1	6.06	22%	7.02	----
BONANZA KING ER #3	USBR	6450	40.5	7.5	19%	8.5	12.8
BIG FLAT D LAKES	USBR	5100	20.0	8.6	43%	9.0	10.6
PETERSON FLAT IN	USBR	6700	33.0	10.4	32%	10.8	11.6
SHIMMY LAKE	USBR	6200	49.9	11.6	23%	12.2	----
RED ROCK MOUNTAIN	USBR	6700	44.0	13.8	31%	14.0	10.6
SACRAMENTO RIVER							
CEDAR PASS	SCS	7100	18.1	10.0	55%	10.3	10.5
BLACKS MOUNTAIN	DWR	7286	8.6	3.4	39%	3.8	----
SAND FLAT	USBR	6750	42.4	14.3	34%	14.6	----
MEDICINE LAKE	USBR	6700	32.7	16.1	49%	16.3	----
ADIN MOUNTAIN	SCS	6350	13.6	7.6	56%	8.0	----
SNOW MOUNTAIN	USBR	5950	27.0	6.7	25%	7.3	9.5
SLATE CREEK	USBR	5600	30.0	7.7	26%	9.1	----
STOUTS MEADOW	USBR	5400	42.5	5.9	14%	6.2	9.1
FEATHER RIVER							
KETTLEROCK	DWR	7300	25.5	11.0	43%	11.6	14.2
GRIZZLY	DWR	6900	29.7	12.0	40%	12.2	13.7
PILOT PEAK	DWR	6800	52.6	8.8	17%	9.7	12.7
GOLD LAKE	DWR	6750	36.5	25.0	68%	25.1	25.6
HUMBUG	DWR	6500	28.0	16.2	58%	16.8	18.7
RATTLESNAKE	DWR	6100	14.0	3.7	27%	4.2	7.4
BUCKS LAKE	DWR	5750	44.7	25.6	57%	26.3	30.5
FOUR TREES	DWR	5150	20.0	10.9	55%	11.2	15.8
YUBA & AMERICAN RIV							

Figure 3: Illustration of a "Table Sorting" behavior. A table is highlighted in the image, and sorted by clicking on the "TODAY" header. Note that highlighting of components (in the case, of matched search term regions) is preserved as the image is manipulated.

4 The Multivalent Document Architecture

We present a brief overview of the multivalent document architecture. A more detailed description can be found in [Phel98].

With layers and behaviors of arbitrary type coming together from multiple sources, a key problem is their coherent composition into a single conceptual document for the user. This integration is accomplished in the multivalent document architecture by several features:

1. A well-defined suite of protocols (implemented as method signatures in Java) to which behaviors should conform. The model's built-in logic promises to compose conforming behaviors.

2. A separation of structural document content from media-dependent elements.

3. A single coherent abstract tree representation of the document, into which all content is combined and upon which all behaviors operate.

4. Support for robust positioning. Mechanisms are provided to robustly align distributed layers, so that it may be possible to position them in the presence of asynchronous changes.

5. An extensible and self-documenting user-interface. Behaviors can add entries to (or create new) conventional pull-down menus, as well as provide help pages for self-documentation.

6. A well-specified format for persistent storage of a multivalent document.

4.1 The MVD Protocol Suite

To allow arbitrary extensibility of any aspect of the system, each of the fundamental runtime operations on digital documents has been opened with an extensible protocol. The fundamental document lifecycle can be found in some (perhaps abbreviated) form in almost all digital document systems. In the MVD model, the life cycle begins with document instantiation (*restore*), i.e., the assembling of components of the document, during which behaviors and layers are loaded, and the behavior methods are inserted into their appropriate places in the other protocols. Then the *build* protocol is started; the *build* methods create an internal graph data structure for the document, using the information in the layers. After *build*, *format* formats the resulting documents, and then *paint* renders the document on the screen. At this point, the *user events* protocol is started. An event loop waits for input from the keyboard, mouse or other input device, and hands it to the methods implementing the protocol. Among other things, events can trigger the *save* protocol, cause the document to *print*, or *select* a portion of the document. In addition, an event might trigger some action that requires looping back to an earlier phase, e.g., to rebuild, reformat, and repaint the document.

Behaviors modify or extend the system by contributing methods to each protocol. Behaviors are ordered, meaning that MVD protocols evaluate methods contributed by behaviors in a specified sequence. In addition, many protocols have “down” and “up” stages, which admits very flexible extension. Given the list of prioritized behaviors loaded into the document, the down phase iterates through all active behaviors from highest priority to lowest, during which the more basic behaviors of higher priority provide a base for lower priority. For instance, in our scanned image example, a high priority behavior establishes the initial document tree based on information from the OCR layer; a lower priority behavior later augment this structure with table description information.

Higher priority behaviors can “short-circuit” ones lower in the chain. For example, as we describe further below, a behavior that “collapses” a segment of a document to achieve an outline mode would, for the sake of efficiency, use short-circuiting to bypass formatting its hidden contents. In the up stage, control flows low priority to high, giving higher priority behaviors the opportunity to “massage” the results from lower priority behaviors. In addition, most protocols have a somwhat complex internal structure. E.g., many protocols are followed while a graph (such as the IDEG, described in the next section) is traversed, and are actually called repeated for each node of the graph; *paint* has some complexities in order to deal efficiently with region geometry, as described below.

MVD supports dynamic reordering of behaviors, meaning that, after a document has been loaded, one can change the order in which behaviors are traversed. However, thus far, we have found dynamic reordering of behaviors to be of limited utility. Specifically, we have only found it useful for the lens behaviors described below. And, in these cases, reordering is done transparently to the user.

4.1.1 The IDEG and Media Adapters

Media of various type (text, video) and format (within text: PostScript, HTML) are encapsulated by specialized behaviors called *media adapters*. During the build stage, these behaviors contribute to the construction of a document structure tree, called the Integrated Document Element Graph (IDEG). The media adapter encapsulates media types, and is responsible for communicating to the MVD infrastructure information such as the bounding boxes of its internal content (at some granularity, e.g., a word or paragraph of a textual document) and for rendering that content upon request from the MVD infrastructure. Separating the document structure from the media elements facilitates a multimedia document system. Behaviors (other than media adapters) operate on the medium-independent structural document tree (i.e., the IDEG) and communicate with encapsulated media types through the protocols. Hence, behaviors can be written once without special accommodation for any particular medium and, as much as it applies to a given medium, operate on all media types.

A behavior's *build* method specifies how that behavior modifies the IDEG. Generally, a document will contain one behavior that builds the primary structure of the IDEG from one or more layers; we refer to this structure informally as the "base" document. Other behaviors' *build* methods may incorporate additional layers into the document structure by modifying the IDEG. In the case of a scanned image document, the primary structure of the document is that obtained by an OCR process. In this case, the IDEG is likely to be simple, just indicating lines and words. The media elements are the individual word images, along with the associated text from the OCR process. In the case of a document in a mark-up language, the structure is likely to mirror that provided by the mark-up. Other behaviors, for example, those that create spans, as described below, will modify the IDEG by marking the beginning and end of the span. The behavior's implementation of the *paint* protocol determines interprets an appropriate user interaction within that span as a command.

Behaviors can contribute items to menus in a conventional menu bar, allowing one form of user interaction. Individual behaviors can also intercept user events, as our table sorting example illustrates.

4.1.2 A Simple Example

Here we give a very simple example of a behavior. Figure 4 contains the entire definition of the "AwkSpan"[1] behavior. This behavior implements a labeled span along the leaves of the IDEG. In general, spans associate some property or functionality with a sequence of media elements.

[1] The name comes from the use of this behavior to put a copyediting comment like "Awkward" next to a span of text.

AwkSpan must (i) attach the span to the IDEG, change the layout of the document so that the label of the span can fit between the lines of text, and (iii) appropriately paint the span and associated label. (Spans are described in some detail in 5.1.1. The span labeled with the text "which?" in Figure 8 below is an instance of AwkSpan.)

```
public class AwkSpan extends Span {
  String awk_;
  Span underspan_;
  Span label_;

  public AwkSpan() {}
  public AwkSpan(Mark l,Mark r, Layer layer, String awk) {
    super(l,r, layer);
    awk_ = awk;
    attach();
  }

  public boolean appearance(Context cx) { cx.underline = layer_.getAnnoColor();
return false; }

  void attach() {
    label_ = new LabelSpan(start.node,start.offset, start.node,start.node.size()
, layer_, awk_);
  }

  public void buildBefore(IdegINode root) {
    super.buildBefore(root);    // attach to tree
    attach();
  }

  public void save(StringBuffer sb) {
    putAttr("COMMENT", java.net.URLEncoder.encode(awk_));
    super.save(sb);
  }
  public boolean restore(ESISNode n, Vector bvect) {
    awk_ = Utility.urldecode(getAttr("COMMENT"));
    return super.restore(n, bvect);
  }

  public void remove() { label_.remove(); super.remove(); }
}
```

Figure 4: The "AwkSpan" behavior.

In terms of interfacing with the MVD protocols, AwkSpan extends *build*, *save*, and *restore*. In a particular, it contributes a method to the "down", or high-to-low stage, of the *build* protocol, i.e., a "buildBefore" method. During this stage, MVD behaviors generally build up the IDEG; in the subsequent low-to-high stage, the minor adjustments to the IDEG might be performed. AwkSpan's buildBefore method simply attaches the span to the IDEG. The contributions to the *save* and *restore* protocols just use system utilities to save the span to and restore it from a hub document.

Almost all the real work for AwkSpan is done elsewhere. In particular, AwkSpan makes use of another behavior, LabelSpan, to deal with formatting and painting. (Several other behaviors use LabelSpan, which exists just to realize code sharing.) LabelSpan is show in Figure 5: The LabelSpan behavior extends the high-to-low stage of the *paint* protocol (i.e., "paintBefore") to properly paint the span's graphic characteristics and its text label.

Despite the fact that AwkSpan (via LabelSpan) must cause changes to the format of the document to allow for the insertion of the span label, no method is contributed to the *format* protocol. Instead, LabelSpan changes the graphics context of the span element (by use of the "appearance" method). The *format* protocol knows to call this function at the appropriate time, thus making the correct formatting information available. Since the primary *format* protocol knows to interpret the graphics context properly, no actual code extension of the protocol is required.

```
public class LabelSpan extends Span {
  String label_;

  public LabelSpan(IdegNode l,int lo, IdegNode r,int ro, Layer layer, String
label) {
    super(l,lo,r,ro, layer);
    label_ = label;
  }

  Font annoFont = new Font("TimesRoman", Font.PLAIN, 10);
  public boolean paintBefore(Graphics g, Context cx, IdegNode start) {
    Rectangle bbox = start.bbox;
    g.setColor(layer_.getAnnoColor()); g.setFont(annoFont);
    g.drawString(label_, cx.x,bbox.y+10/*2 points of descender*/);
    return false;
  }

  public boolean appearance(Context cx) { cx.spaceabove =
Math.max(cx.spaceabove,12); return false; }
}
```

Figure 5: The LabelSpan behavior

While we have not described all the details of this simple behavior, the example illustrates rather typical behavior construction. Specifically, there are typically a number of behaviors that implement a particular function, for the sake of being able to write modular code. There may be arbitrary functions that exist as part of the behavior, but they must ultimately interface with the protocols. Sometimes what the behavior wants to do is anticipated by the infrastructure (as in the case of *format* understanding how to lay out elements with certain graphics properties), so a behavior may alter a data structure rather than extend a protocol.

4.2 Robust Anchoring

MVD assumes that documents are composed out of distributed layers. These are likely to be under the control of different authorities. Rather than attempting to impose a requirement of strict coherence (referential integrity), which we feel is untenable given our openness goal, we provide support for redundant descriptions of positions across layers. Specifically, a standard system class is provided that takes a document structure tree position and creates a redundant description of that place. This description includes (i) the place's structural position in the IDEG (similar to HyTime's [DD94] TREELOC), along with any media type-specific offset into the leaf node, (ii) an excerpt of the underlying text (if any), and (iii) a unique identifier. If the document is restored at a later time with the base document or other layers upon which it depends edited, a series of incrementally permissive back-off strategies tries to reattach the anchor to the new appropriate location.

For example, if a block of text were deleted before an anchor, the structural tree position may be invalid, but the text will be searched for and, if the excerpted text is unique in the document, the anchor will be placed at the match. If the corresponding text were edited as well, we search for smaller and smaller portions of the text down to some minimum length until a match is possible. Closer matches to the original location are preferred to those farther away when several matches produce a choice.

Preliminary results indicate that in practice this anchor repositioning strategy works well. Of 754 annotations using such anchors that needed repositioning to layers that underwent varying degrees of mutation, 742 were automatically repositioned, leaving 12 to be reapplied by the user. In most of the latter cases, the associated position had in fact been deleted entirely.

While robust anchoring is provided by the MVD infrastructure itself, we note that it could have been implemented as an ordinary behavior. It is such a key service, however, that rather than relying on a hub document that may or may not include it, we decided to include robust anchoring as part of the core set of document services upon which all behaviors may depend. However, behavior implementers might provide additional functionality to deal with asynchronous changes. For example, if every attempt at reattachment fails, as when the corresponding area of the document is deleted entirely, this fact could be reported to the user, who can reattach the object manually or discard it. Or, the original author could be alerted to the change, and perform an update if desired.

4.3 The User Interface

The MVD model poses interesting user interface issues, as new behaviors can add arbitrary functionality. Also, one of our user interface design goals is to allow the user to interact with the application without necessarily understanding all the details of the model. While we regard our current user interface capabilities as preliminary, we believe we have at least partially addressed these problems.

First, we have provided a general framework into which behaviors can interact with the user. Specifically, each behavior can add one or more entries to be assembled into conventional pull-down menu entries. Entries can be specified as belonging to functional groups that are mutually exclusive or independent, so that the menus can be arranged accordingly. The detailed mapping of behavior menu specifications to actual menu entries is specified in the hub document, allowing more flexibility (and more potential user confusion if care is not exerted). For convenience, a default assembly of groups into menu items is provided in the infrastructure, so that if no specification is made, the menu item will probably end up in a meaningful location.

In addition, each behavior can provide help pages that document its function. One way to access this information is on the “Help on Menu Item” entry in the “Help” menu. After this menu entry is selected, the cursor changes, and the next selection of a menu item will bring up a page describing the effect of that menu entry, the behavior to which it belongs, and perhaps other related menu items. (Of course, this feature is provided via a help behavior, not by the infrastructure per se. In general, we imagine that a number of generally useful behaviors will be packaged and included in most documents, but the infrastructure does not require their inclusion.)

Some other forms of help are provided on a per-behavior basis, but along the lines of suggested conventions. For example, behaviors can offer information about regions of the document to which they pertain. They do so generally by providing a menu item in the “Meta” menu. For example, the behavior managing image-OCR integration will show where it thinks word boundaries are via such a method. The user can also dynamically load and unload behaviors. Again, this is done not as part of the infrastructure, but via a “behavior loading behavior”, itself initially loaded under the File menu. Similarly, another behavior describes which behaviors are loaded and what they do.

4.4 The Hub Document

As mentioned above, the persistent form of a multivalent document is the “hub” document. The hub document is essentially a list (written in the language XML [BS97]) of the layers and behaviors that comprise the single conceptual document. Figure 6 shows a slightly abbreviated example of a hub document, in this case, for the scanned image example of the previous section.

In Figure 6, behaviors available for all pages are listed first. (Only the TableSorting behavior is restricted to a single page.) The layers utilized by the behavior XDOC, the media adapter for scanned page images, are given as URLs. Other data layers, e.g., the text of a note, is given in line. Behaviors can choose whether to incorporate their associated layers by reference or inline, with large layers, non-textual media, and layers meant to be shared across documents more naturally included by reference. In this example, the behaviors will be loaded from a default location on the document server. Similarly, no specification of mappings from menu groups to menu locations is given, so the default configuration is used.

As part of the listing of the behaviors to be included in a document, the hub document may also include behavior instances, i.e., specific uses of a behavior within a document. It is through the use of such behavior instances that most annotation is effected in MVD. We will discuss behavior instances more below.

To open a document, the framework fetches the behaviors specified in the hub document, and the layers they reference, and composes them together. I.e., it places the methods specified by the behaviors into the appropriate protocols, and begins following the protocols. The order of behaviors in the hub determines the initial relative priority of the behaviors. It is primarily of significance that certain fundamental behaviors be loaded first

The framework can write out a new hub document (and, possibly, new or updated layers as well). The new hub document may reflect various changes that occurred during the course of using MVD, including behavior instances created or eliminated or changes to a layer. Generally, it is the job of each behavior's *save* protocol implementation to write out the portions of a hub document that reflect the current status of behavior instances or changes to a layer.

In this example, the layers are primarily images and OCR rendered from those images. Another example of a hub document is a "base" document along with layers and behaviors that impose annotations on this document. We examine a variety of such examples below.

```
<Multivalent title="Testing" author="Authoritative" source="Source">
<Pages Behavior=Multivalent.Pages page=1 pagecnt=20>
<Xdoc Behavior=Multivalent.Xdoc XdocURL="http://elib.cs.berkeley.edu/docs/data/0000/17/OCR-
XDOC/%08s.xdc" ImageURL=" http://elib.cs.berkeley.edu/docs/data/0000/17/GIF-
INLINE/%08s.gif" Scale=3.32>
</Xdoc>
<Page number=8>
<TableSort Behavior=Multivalent.TableSort>
…
</TableSort>
</Page>
<Doublespace Behavior=Multivalent.Doublespace active=off></Doublespace>
<Hyperlink Behavior=Multivalent.Links></Hyperlink>
<Search Behavior=Multivalent.Search Active=off></Search>
<Magnify Behavior=Multivalent.Magnify active=off x=300 y=200></Magnify>
<ShowOCR TITLE="OCR Lens" Behavior=Multivalent.ShowOCR active=off x=50 y=200 width=300
height=300></ShowOCR>
<ShowImage TITLE="Scanned Image Lens" Behavior=Multivalent.ShowImage active=off x=100 y=200
width=300 height=300></ShowImage>
<Note Behavior=Multivalent.Note x=200 y=50 active=off>Page 8 illustrates\nthe tablesorting
behavior.\nClick on a column title.</Note>
<ShowOCR TITLE="OCR Lens" Behavior=Multivalent.ShowOCR active=off x=50 y=200 width=300
height=300></ShowOCR>
<High Behavior=Multivalent.HighMan></High>
<SelProvenance Behavior=Multivalent.SelProvenance></SelProvenance>
<SelMarkup Behavior=Multivalent.SelMarkup active=off></SelMarkup>
<Debug Behavior=Multivalent.Debug></Debug>
<Bed Behavior=Multivalent.Bed></Bed>
</Pages>
</Multivalent>
```

Figure 6 : A hub document. Shown in abbreviate form is the hub document for the scanned image document used in the previous figures.

While this description of the architecture is very brief, we hope it at least suggests how the model can be highly distributed (the layers and behaviors, specified by a hub document, can reside anywhere, and are composed robustly by the infrastructure), highly open (the layers can be of varying data types, provided that a media adapter behavior is made available for that type, and the document can be viewed and manipulated on any Java-compliant platform), and highly extensible (all functionality is opened to extension of the protocols).

5 Multivalent Annotations

Above we proposed a number of criteria that an adequate model of annotation should meet. The multivalent document model outlined provides a straightforward means to meet these criteria. Specifically, one can construct a multivalent document that includes as a layer a "base" document, i.e., the document to be annotated, and also, some behaviors and associated data that effect the desired annotations. "Multivalent annotations" are then just MVD behavior instances used annotatively. Moreover, if the base document is created by one author, and the annotative behavior instances by another, then the result is a distributed annotation.

While there is nothing built in to the MVD model in support of annotation per se, some behaviors can naturally be used to effect annotative functions. In addition, combinations of such behaviors can achieve coherent, recognizable genres of annotation. In this section, we discuss some types of MVD behaviors, and illustrate their annotative applications.

5.1 Types of Behaviors

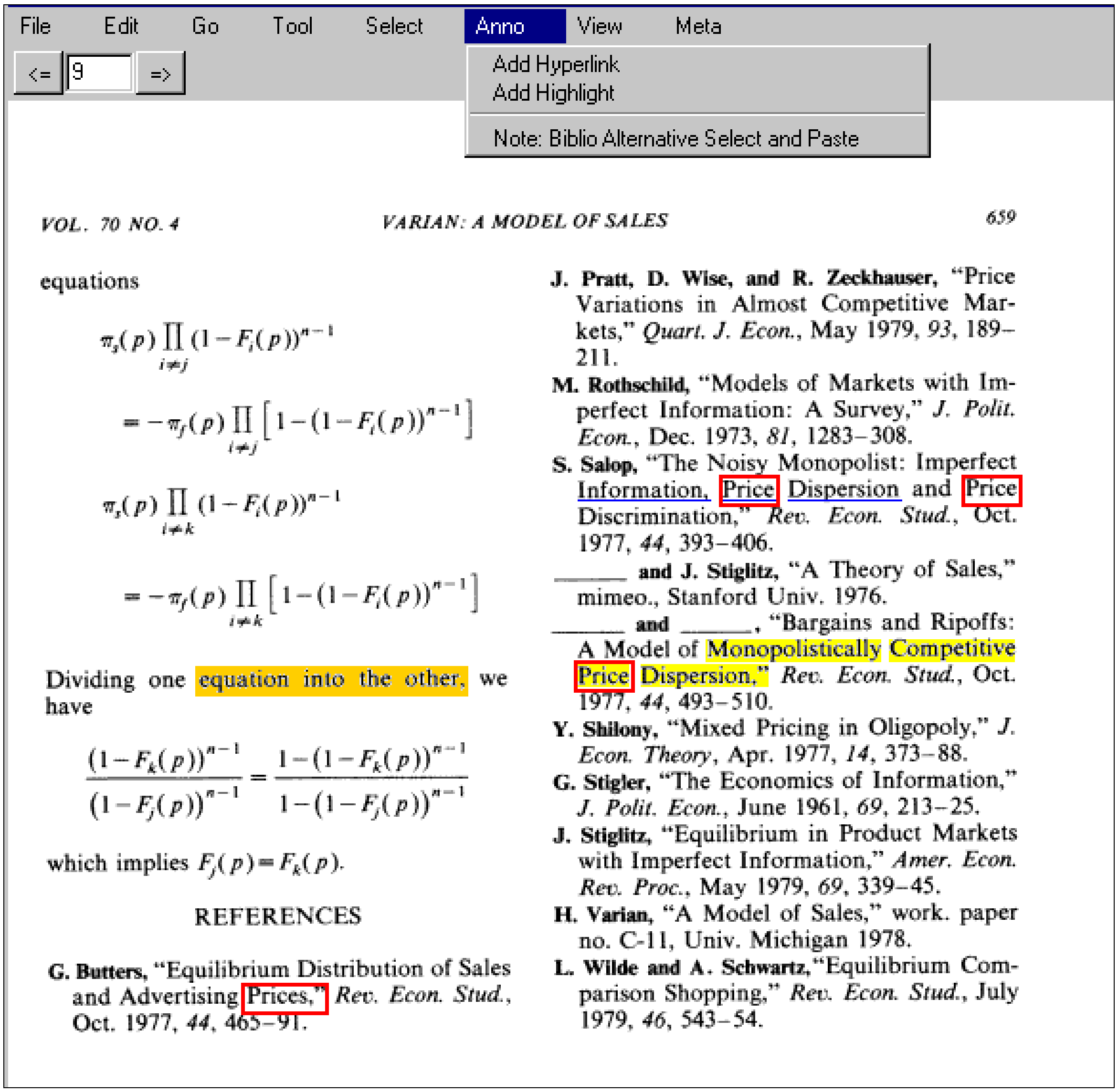
We have developed a number of multivalent behaviors that have annotative (as well as other) uses. We find it convenient to divide these behaviors into three classes, as categorized by their underlying technologies. There are behaviors that make use of (1) point-to-point *spans* of media elements, (2) *geometric regions* of a document presentation, and (3) *structures* within the document tree. Here we give examples of each class, and illustrate how they can be used annotatively. We also show some examples of uses of these behaviors in coherent combinations.

As behaviors are in effect extensions of the MVD framework, the functionality we illustrate here is not part of the framework itself. However, the behaviors used in these types have emerged as important to our model's goals, and hence we have designed the infrastructure, if not to support such behaviors directly, to provide efficient mechanisms that these behaviors require. We reiterate that the model is not limited to behaviors (and hence, modes of annotation) fitting into these categories.

5.1.1 Spans

Span are behavior instances that extend from one point in the document continuously to another point. Spans are implemented as behavior instances with (robustly specified) start and end points in the document tree. As a span is just a behavior instance, there is generally one behavior corresponding to each span type. That behavior can create new spans, influence how these appear, handle user interactions, and save and restore spans from persistent storage. Spans generally modify the current display parameters (the graphics context) before the content itself is drawn, and can receive user events such as mouse clicks and keypresses.

Figure 7 shows a document containing several types of spans. This document is a scanned page image, enlivened by an optical character recognition layer and a set of supporting behaviors, as described in the previous section. One type of span illustrated in the example is a hyperlink. It is presented by having the associated document span be underscored in blue. It has user interface properties one has come to expect from hyperlinks, in that moving the cursor over the link changes the cursor to a pointer, and reveals a destination at the bottom of the page. Clicking on the link accesses the resource being referred to.



VOL. 70 NO. 4 VARIAN: A MODEL OF SALES 659

equations

$$\pi_s(p)\prod_{i\neq j}(1-F_i(p))^{n-1} = -\pi_j(p)\prod_{i\neq j}\left[1-(1-F_i(p))^{n-1}\right]$$

$$\pi_s(p)\prod_{i\neq k}(1-F_i(p))^{n-1} = -\pi_j(p)\prod_{i\neq k}\left[1-(1-F_i(p))^{n-1}\right]$$

Dividing one equation into the other, we have

$$\frac{(1-F_k(p))^{n-1}}{(1-F_j(p))^{n-1}} = \frac{1-(1-F_k(p))^{n-1}}{1-(1-F_j(p))^{n-1}}$$

which implies $F_j(p)=F_k(p)$.

REFERENCES

G. Butters, "Equilibrium Distribution of Sales and Advertising Prices," *Rev. Econ. Stud.*, Oct. 1977, *44*, 465–91.

J. Pratt, D. Wise, and R. Zeckhauser, "Price Variations in Almost Competitive Markets," *Quart. J. Econ.*, May 1979, *93*, 189–211.

M. Rothschild, "Models of Markets with Imperfect Information: A Survey," *J. Polit. Econ.*, Dec. 1973, *81*, 1283–308.

S. Salop, "The Noisy Monopolist: Imperfect Information, Price Dispersion and Price Discrimination," *Rev. Econ. Stud.*, Oct. 1977, *44*, 393–406.

______ and J. Stiglitz, "A Theory of Sales," mimeo., Stanford Univ. 1976.

______ and ______, "Bargains and Ripoffs: A Model of Monopolistically Competitive Price Dispersion," *Rev. Econ. Stud.*, Oct. 1977, *44*, 493–510.

Y. Shilony, "Mixed Pricing in Oligopoly," *J. Econ. Theory*, Apr. 1977, *14*, 373–88.

G. Stigler, "The Economics of Information," *J. Polit. Econ.*, June 1961, *69*, 213–25.

J. Stiglitz, "Equilibrium in Product Markets with Imperfect Information," *Amer. Econ. Rev. Proc.*, May 1979, *69*, 339–45.

H. Varian, "A Model of Sales," work. paper no. C-11, Univ. Michigan 1978.

L. Wilde and A. Schwartz, "Equilibrium Comparison Shopping," *Rev. Econ. Stud.*, July 1979, *46*, 543–54.

Figure 7: An image document with some spans. The spans shown here include a hyperlink (presented by the blue underscore), and a highlight (yellow background). Spans corresponding to the current selection (orange background) and search results (red boxes) are also shown. A menu (Anno) has been pulled down, revealing entries for creating hyperlink and highlight spans. If either is now chosen, the current selection span with become an additional hyperlink or highlight span.

Another span on this page is a highlight, meant to resemble a yellow marker pen. In addition, the results of a search are also displayed by spans marked by red boxes. The current selection (display with an orange background) is also implemented as a span.

Note that in this figure, as in all our examples, it is not possible to discern exactly how these spans relate conceptually to the underlying image. Indeed, three possible scenarios are worth distinguishing. In one scenario, these spans are described in the hub document created by the author of the underlying image document. In this case, the author has used MVD capabilities to enrich his or her own document, perhaps adding features to it beyond those the base document format can support. However, the annotations and images are likely to live on the same server, and hence, even if we were to construe them as annotations by the author, they would not be taking advantage of the distributed nature of the model. A second, more interesting, scenario, is one in which these annotations are described in a hub document created by another author, and which resides on a server other than the one holding the image. In this case, the spans comprise distributed annotations of the original document.

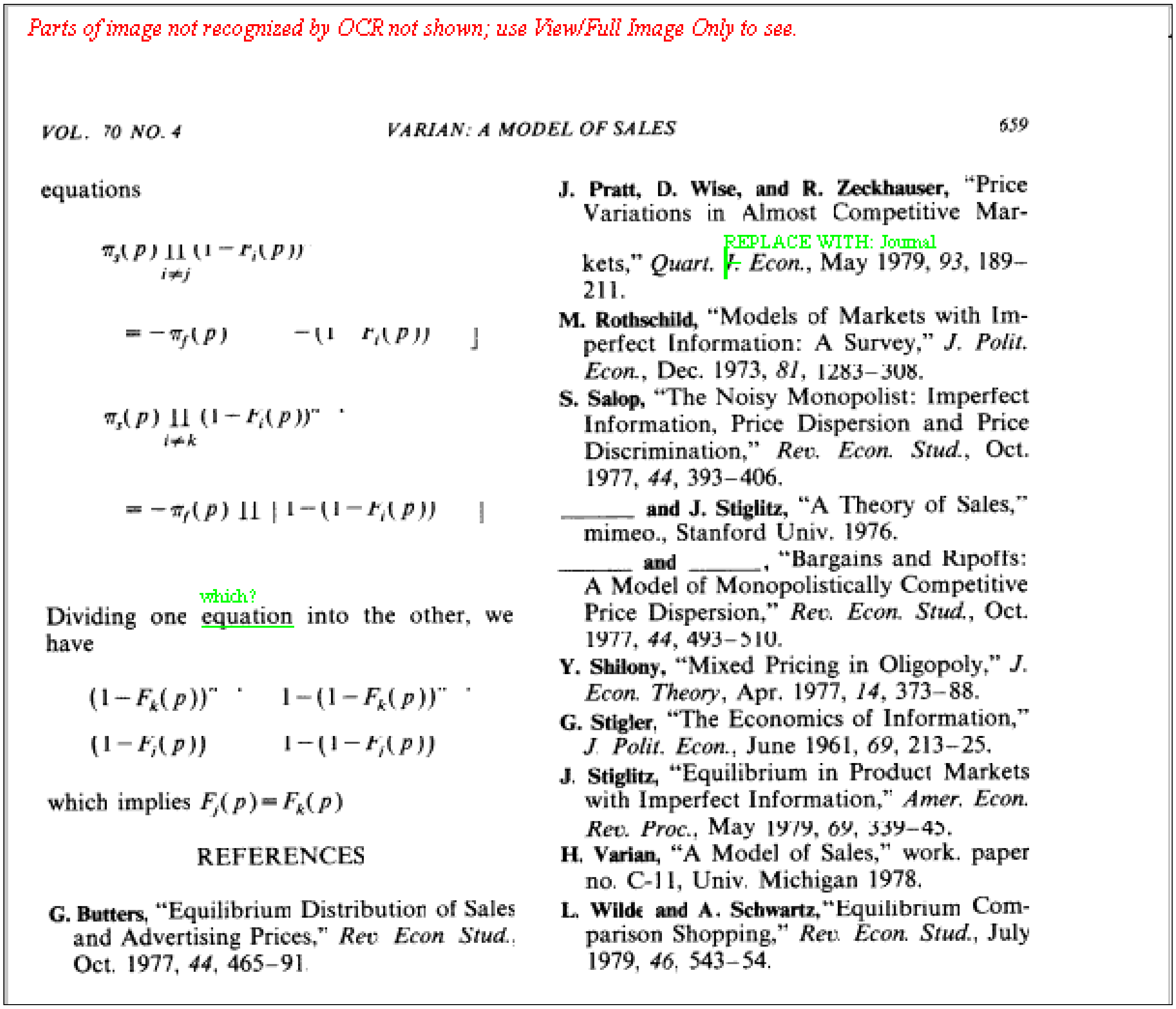
In a third scenario, the spans have not (yet) been stored anywhere, but instead, have been created in the MVD client's internal representation of the document, perhaps just for the user's temporary purposes. This is certainly the case for the current selection, and most likely (but not necessarily) for the search result. However, the user may subsequently save the spans in a new hub document, thus created some annotations of the base document. For example, in the figure, the pulled down "Anno" menu reveals entries for creating highlight and hyperlink spans. Should one of these menu entries now be selected, the current highlight span would become a highlight or hyperlink span. The user may then decide to save the new span by writing out a new hub document, hence creating a persistent, and potentially distributed, annotation on the base document.

Of course, any combination of these scenarios is possible. E.g., the author might have added the hyperlink as part of the initial publication. An annotator may have added the highlight (i.e., used MVD to create the highlight, and then to create another hub document containing this span persistently). A third user may have opened the document created by the second user, and done a search, resulting in the search terms being outlined.

Before continuing, it is useful to clarify the relation of behaviors and behavior instances, such as spans. In this example, there is one hyperlink behavior, and one highlight behavior. Each of these behaviors may create and manage any number of spans, i.e., it may have any number of instances. The behaviors themselves may indicate that they want an entry in a menu, the use of which might create a new behavior instance, or the like. The behaviors' *user events* methods intercept events on these spans; their *paint* methods set the graphics properties for the span, and so on.

Another class of spans we have implemented are copyediting marks. Figure 8 is an example of the same document, but with copyediting behaviors loaded, and used to create a few spans. One span, in the left hand column, extends over the word "equation". This span simply provides a comment. The second span, on the upper right, specifies replacement text. In this case, the copyediting mark extends through the word "*J.*", and recommends that this be replaced with the text "Journal".

Note that these spans illustrate a powerful feature of the MVD infrastructure, namely, layout control. These spans effectively change the height of associated lines, and image management behavior must re-layout the document to accommodate them. It does so by painting the document from its OCR rendering. Since such rendering is imperfect, a warning is displayed to this effect. The user can use menu entries to toggle back and forth between these (and other) presentations of the document. (Note that commercial OCR engines, such as the one used here, generally do a very poor job with equations, so that that region of the document is redrawn particularly poorly.)



Parts of image not recognized by OCR not shown; use View/Full Image Only to see.

VOL. 70 NO. 4 VARIAN: A MODEL OF SALES 659

equations

$$\pi_s(p) \prod_{i \neq j} (1 - F_i(p))$$

$$= -\pi_f(p) \quad -(1 \quad F_i(p)) \quad]$$

$$\pi_s(p) \prod_{i \neq k} (1 - F_i(p))^{n}$$

$$= -\pi_f(p) \prod | 1 - (1 - F_i(p)) \quad |$$

Dividing one equation into the other, we have

which?

$$\frac{(1 - F_k(p))^{n}}{(1 - F_j(p))} \quad \frac{1 - (1 - F_k(p))^{n}}{1 - (1 - F_j(p))}$$

which implies $F_j(p) = F_k(p)$

REFERENCES

G. Butters, "Equilibrium Distribution of Sales and Advertising Prices," *Rev Econ Stud.*, Oct. 1977, *44*, 465–91.

J. Pratt, D. Wise, and R. Zeckhauser, "Price Variations in Almost Competitive Markets," *Quart. F. Econ.*, May 1979, *93*, 189–211.

REPLACE WITH: Journal

M. Rothschild, "Models of Markets with Imperfect Information: A Survey," *J. Polit. Econ.*, Dec. 1973, *81*, 1283–308.

S. Salop, "The Noisy Monopolist: Imperfect Information, Price Dispersion and Price Discrimination," *Rev. Econ. Stud.*, Oct. 1977, *44*, 393–406.

______ and J. Stiglitz, "A Theory of Sales," mimeo., Stanford Univ. 1976.

______ and ______, "Bargains and Ripoffs: A Model of Monopolistically Competitive Price Dispersion," *Rev. Econ. Stud.*, Oct. 1977, *44*, 493–510.

Y. Shilony, "Mixed Pricing in Oligopoly," *J. Econ. Theory*, Apr. 1977, *14*, 373–88.

G. Stigler, "The Economics of Information," *J. Polit. Econ.*, June 1961, *69*, 213–25.

J. Stiglitz, "Equilibrium in Product Markets with Imperfect Information," *Amer. Econ. Rev. Proc.*, May 1979, *69*, 339–45.

H. Varian, "A Model of Sales," work. paper no. C-11, Univ. Michigan 1978.

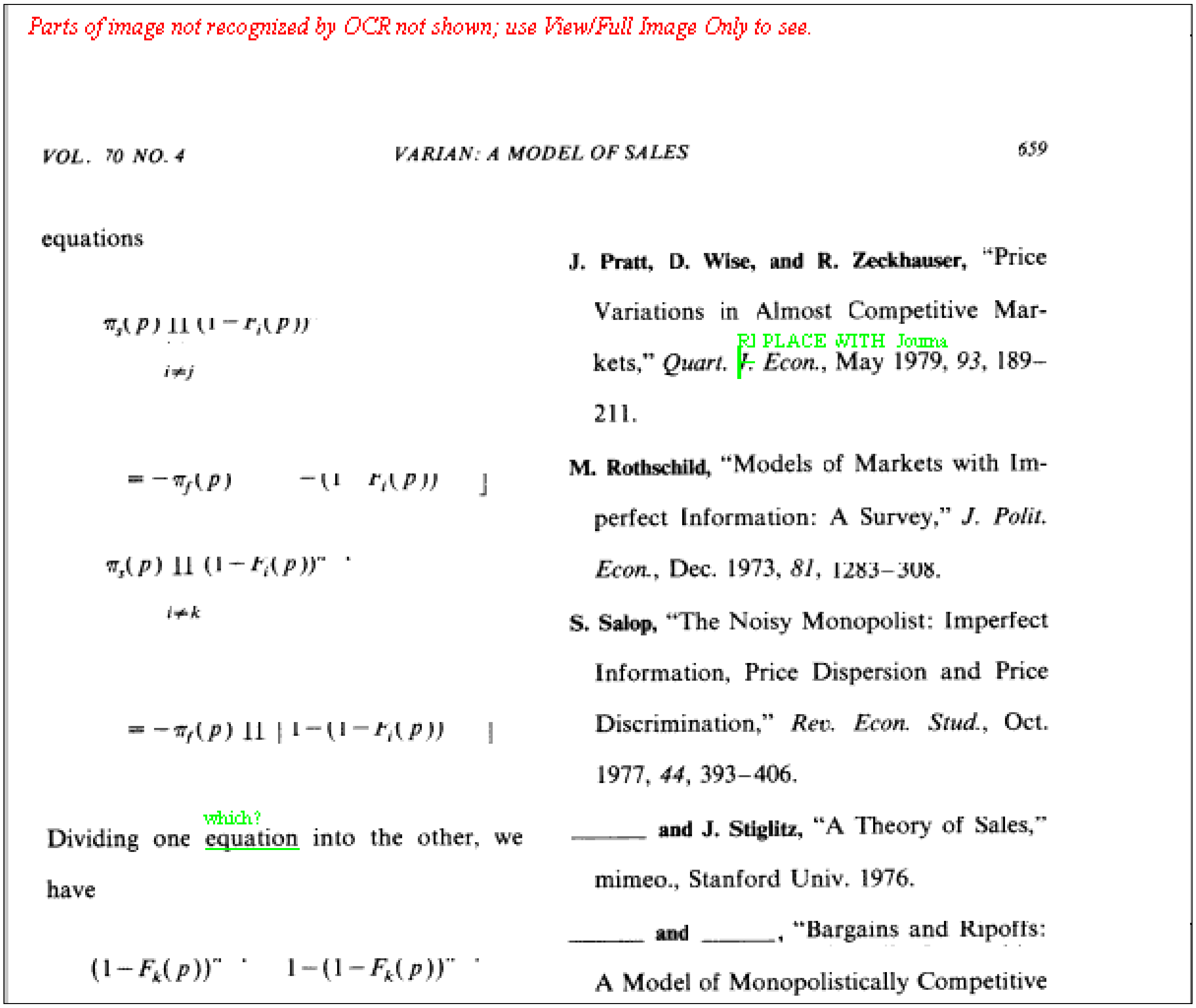
L. Wilde and A. Schwartz, "Equilibrium Comparison Shopping," *Rev. Econ. Stud.*, July 1979, *46*, 543–54.

Figure 8: Copyediting spans on a scanned image

There are several advantages to using spans of this sort for annotations. Unlike simple overlays, spans are anchored to document contents, so the annotation will be coherent after additional document manipulation. For example, Figure 9 shows the same document in which layout control has been used to double-space the page image (by the use of a doublespace behavior). Note that the copyeditor marks have stayed with the appropriate text. Had this annotation been achieved by a simple bit image overlay (which one can also easy accomplish in the MVD framework), the contents of the overlay would no longer be aligned with the base after such a manipulation.[2] In addition, spans can be made robust, so that they still may be positioned correctly in the presence of change in the underlying base document.

---

[2] Of course, one might want to have spans whose annotative content is a bit image, rather than a structured object as in these examples. A behavior supporting such spans should be easily accommodated within the MVD framework.



Parts of image not recognized by OCR not shown; use View/Full Image Only to see.

VOL. 70 NO. 4 VARIAN: A MODEL OF SALES 659

equations

$$\pi_s(p) \prod_{i \neq j} (1 - F_i(p))$$

$$= -\pi_f(p) \quad -(1 \quad F_i(p)) \quad]$$

$$\pi_s(p) \prod_{i \neq k} (1 - F_i(p))^n$$

$$= -\pi_f(p) \prod | 1 - (1 - F_i(p)) \quad |$$

Dividing one equation into the other, we have

which?

$$(1 - F_k(p))^n \quad 1 - (1 - F_k(p))^n$$

J. Pratt, D. Wise, and R. Zeckhauser, "Price Variations in Almost Competitive Markets," Quart. J. Econ., May 1979, 93, 189–211.

RJ PLACE WITH Journa

M. Rothschild, "Models of Markets with Imperfect Information: A Survey," J. Polit. Econ., Dec. 1973, 81, 1283–308.

S. Salop, "The Noisy Monopolist: Imperfect Information, Price Dispersion and Price Discrimination," Rev. Econ. Stud., Oct. 1977, 44, 393–406.

______ and J. Stiglitz, "A Theory of Sales," mimeo., Stanford Univ. 1976.

______ and ______, "Bargains and Ripoffs: A Model of Monopolistically Competitive

Figure 9: A scanned image double-spaced. The copyedit marks remain coherent.

This type of annotation is more interesting when the base document is more readily subject to manipulation. As an example, consider Figure 10, in which the base layer is one of the authors home page on the web page, i.e., an HTML document. Here we have added four copyediting marks: Suggestions that "Acrobat" be replaced by "PDF"; that the word "that" four lines below be deleted; that "Roget's Thesaurus, Fifth Edition", be italicized; and that a missing parenthesis be added at the end of a line.

This example illustrates a number of points. First, it demonstrates support for HTML within the framework. This was done simply by writing a media adapter for this document format. Second, the same behaviors are seen operating on multiple, rather different, document formats: scanned images above, and HTML here. (We have explicitly illustrated copyeditor marks operating on two document formats; all the other behaviors shown above, e.g., highlighting and searching, also operate across formats. Indeed, HTML hyperlinks (many of which appear in the example) are implemented in MVD by turning the underlying HTML markup into MVD hyperlink spans.)

Figure 10: Copyediting marks on an HTML document. The copymarks appear in a blue-green font, and suggest, in order, the following actions: replacing the word "Acrobat" with "PDF"; deleting a gratuitous "that" (about the middle of the page); italicizing the words "Roget's Thesaurus, Fifth Edition", and inserting an omitted parenthesis.

Third, because MVD annotations are supported by behaviors, they can have useful functionality. Copyeditor spans, for example, are executable: Moving the cursor over a copyeditor span and clicking carries out the advertised action. For example clicking on the deletion and italicize spans in the current example produces the result shown in Figure 11.

Executable copyediting marks illustrates the separation between document structure and media type in MVD. The copyediting spans refer to leaves in the MVD IDEG. The IDEG is essentially the same for documents created from scanned images or HTML, although in the former case the IDEG is created from the (generally simple) structure given to us by an OCR process, and in the latter, from the parse tree for an HTML document. However, the leaves of the IDEG for a scanned image refer to image regions; the leaves of the IDEG for HTML contain strings of text. The latter are relatively easy to manipulate, the former much harder. For example, it is relatively easy for a behavior to implement "italicize" for text, as doing so corresponds to setting a graphics property. If we wanted this same copyediting behavior to be executable in images, one would have to implement a transformation specific to that media type. Doing so would require a considerable amount of effort, for marginal utility, so we have not done so. Of course, someone believing that this functionality would be valuable can implement such behaviors within the framework.

We will return to this example below, in the discussion of Notemarks, where we will demonstrate additional ways of taking advantage of capabilities afforded by spans.

Figure 11: The HTML page after executing two copyediting marks, namely, a deletion and an italicization.

Before we leave spans, we note that we have only illustrated a few types here. We have also written behaviors can change most graphics properties, including background color, foreground color, underline/overstrike, font, scaling factor, visibility, and so forth, i.e., the general range of textual attributes found in most document systems.

Also, while we have shown some uses of spans, e.g., highlights and hyperlinks, that are familiar to users of various word processing and hypertext systems, here we have shown how the MVD model allows these notions to be imposed on a document format (scanned images) that does not support such capabilities per se. We have also shown how such behaviors, while not intrinsically annotational, can be used annotatively, by storing their instances in a hub document separate from the underlying base document. (Note that this annotative capability may be useful for document formats that do support these types internally. For example, one could add a new hyperlink to someone else's web page.) We have also demonstrated annotation *in situ*. Finally, span annotations are robust against changes to document content and against document manipulation. As described above, when saved to persistent storage, spans use a system class to achieve gracefully degrading repositioning. (Internally, spans are anchoring with "sticky pointers" [Lad], which maintain perfect alignment if a layer changes during a session.) Span annotations are deleted when the entire span being annotated is deleted, and expand to cover any text inserted into the span.

5.1.2 Geometric Region Behaviors: Lenses

Spans allow reference to the fine-grain structure of a document. Another type of multivalent behavior, lenses, affect geometric regions of a document's appearance. MVD lenses were inspired by Xerox PARC's Magic Lenses [Bier93]. Like spans, MVD lenses can modify content display parameters before document content is drawn and can receive events. As such, we were able to implement lenses within the multivalent model with no specific allowance for them; they were implementable as behaviors simply by using the fundamental operations on digital documents exposed by the MVD protocols. However, it was not possible to compose lenses efficiently in this matter, so support for lenses was introduced directly into the core of the MVD framework.

Figure 12 illustrates two examples of lenses. Toward the upper left portion of the screen is a "Show OCR" lens. Inside this lens, the image text is replaced by the results of an OCR process, rendered in the font the OCR software estimates for the original text. (As mentioned above, and can be seen in this example, the OCR software does a much more credible job with text than with mathematics.) Toward the lower right portion of the screen is a "Bit Magnify" lens. This lens enlarges the image underneath it. Where the lenses overlap, the effects compose: The upper left portion of the "Bit Magnify" lens overlaps the "Show OCR" lens, and so enlarges the OCR translation; the rest of the lens enlarges the base scanned image. Of course, lenses compose with other types of behaviors as well. Where lenses overlap the selected span, or the copy-editor marks, that element composes with the effect of the lens. User interactions also compose through the lenses, so, for example, one can select text or invoke hyperlinks through the lens and so forth.

Lenses compose according to their order in the protocol hierarchy. The user can reorder the relative priority of behaviors simply by clicking on a lens's title bar, bringing that lens to the top of the stack. (This is in fact the only case we have encountered so far in which it is useful and intuitive for the user to change the order of behaviors in the protocol.)

Lenses can be moved about by their title bar, resized by dragging the lower right corner, and removed by clicking the close box at the right of the title bar. Lenses also receive events, which they can block, let pass through, or transform. For example, the magnify lens adjusts the x,y coordinates of mouse cursor positions to correspond to the underlying appearance, enabling selection or hyperlink activation inside.

Lenses can arbitrarily transform the appearance of their contents. Some transformations can be accomplished simply by altering a graphics property of the region. Other transformations, e.g., that done by "Show OCR, require the lens to carry a list of attribute-value pairs, e.g., "show—OCR" (as opposed to "show—image"), which receptive media elements know to look for in order to draw themselves appropriately. In composing overlapping lenses, conflicting settings between two lenses are overridden by the higher priority lens, i.e., the one that appears "on top".

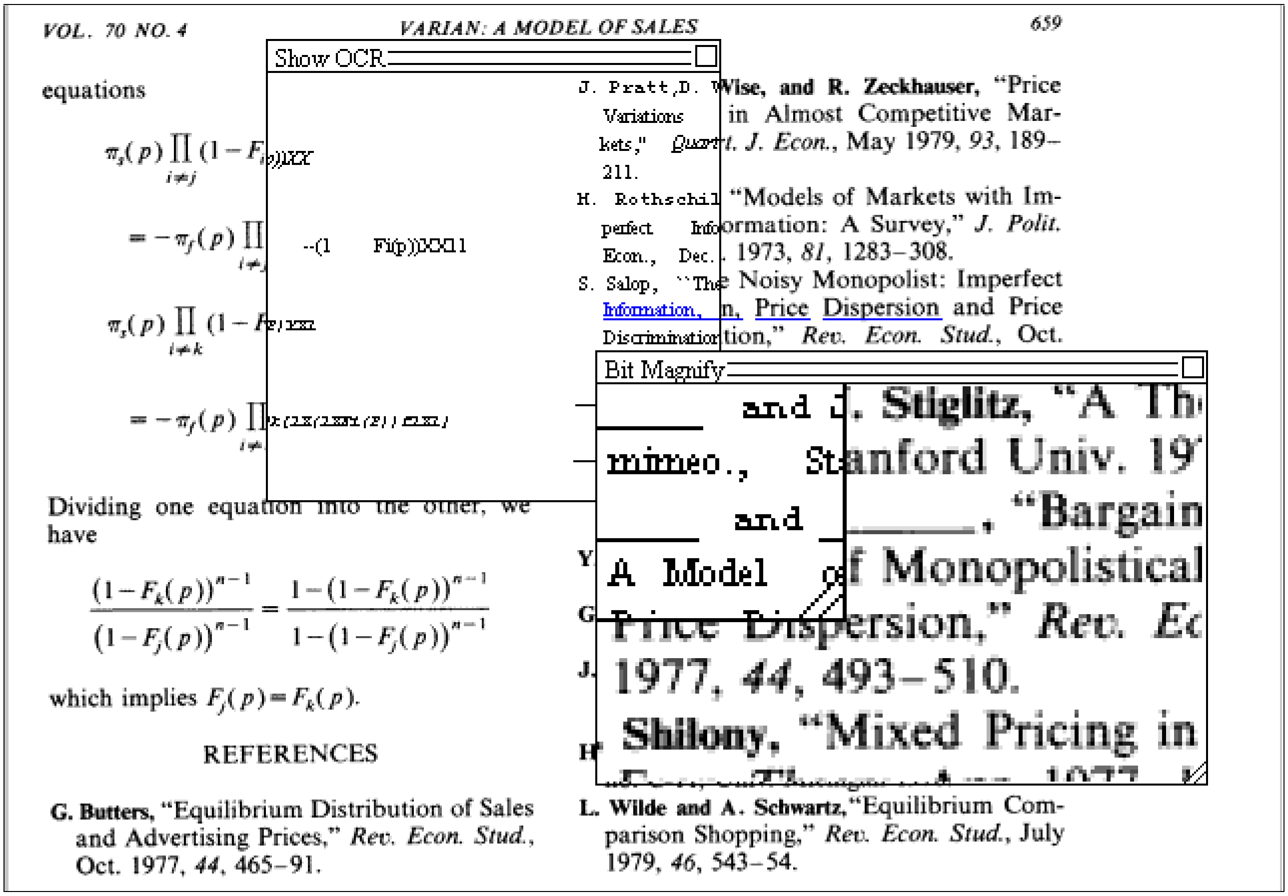


Figure 12: Geometric or lens behaviors: One "Show OCR" and one "Bit Magnify" lens are shown, in composition in overlapped regions.

It is certainly possible to use geometric region behaviors annotatively. E.g., one might use a magnify lens to draw attention to a detail of the document being annotated. However, the most obviously annotative use of lenses is the Post-it™-style note. Such notes are implemented as "opaque" lenses. A note is simply a lens which, during its down (i.e., high-to-low) priority phase of painting, paints the background and then short-circuits any lenses below. Since a note lens shares the window apparatus with other types of lenses, the note can be moved about the page and resized.

Notes can contain their own document contents. Such context can be an IDEG, and therefore, make its own use of behaviors. One particularly useful example is shown in Figure 13. Here we show the top of a relatively long HTML page, annotated with a note whose contents include a hyperlink to a location further down the page, where additional annotations have been placed. The anchor for this reference was also added to the document externally. (Anchors are implemented as a type of span.) Now, the reader need only click on the link to be transported to an off-screen comment. (We also included a Bit Magnify lens in this example, to illustrate its annotative use, and to illustrate its functioning on another document format.)

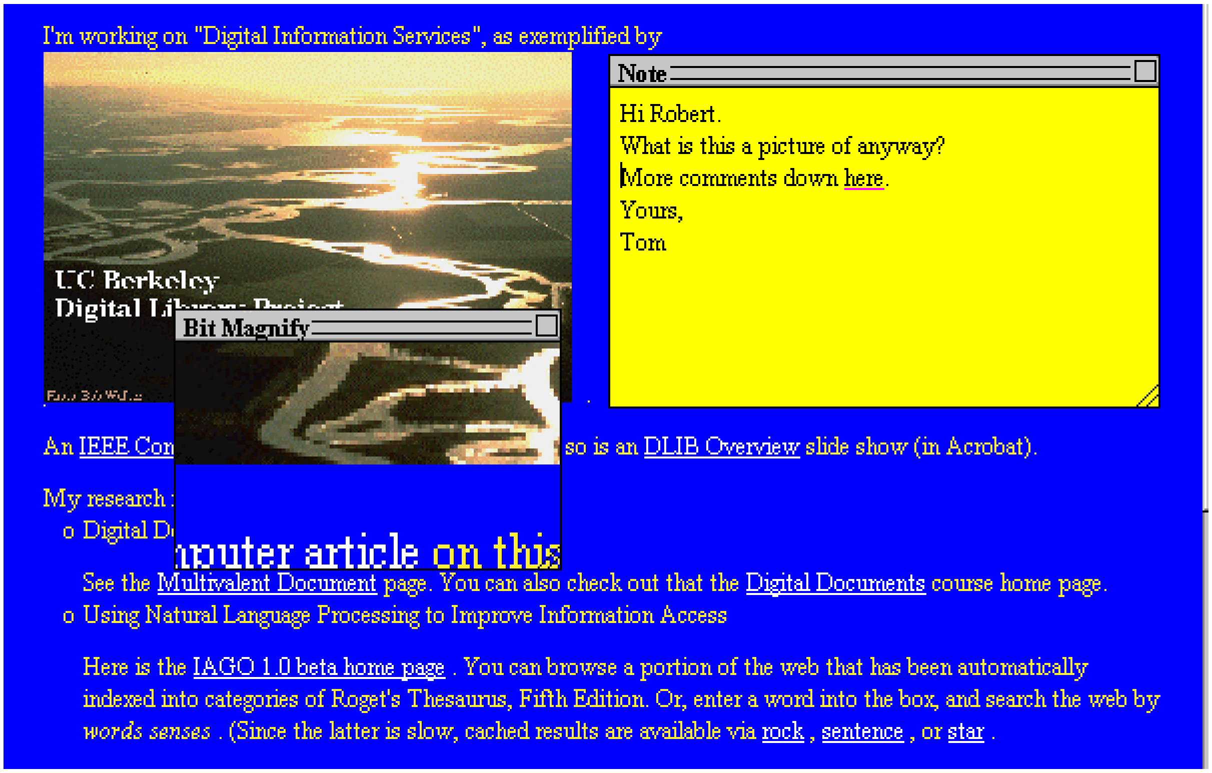


Figure 13: A note and an magnify lens on an HTML document. The note contains a hyperlink to additional annotations offscreen.

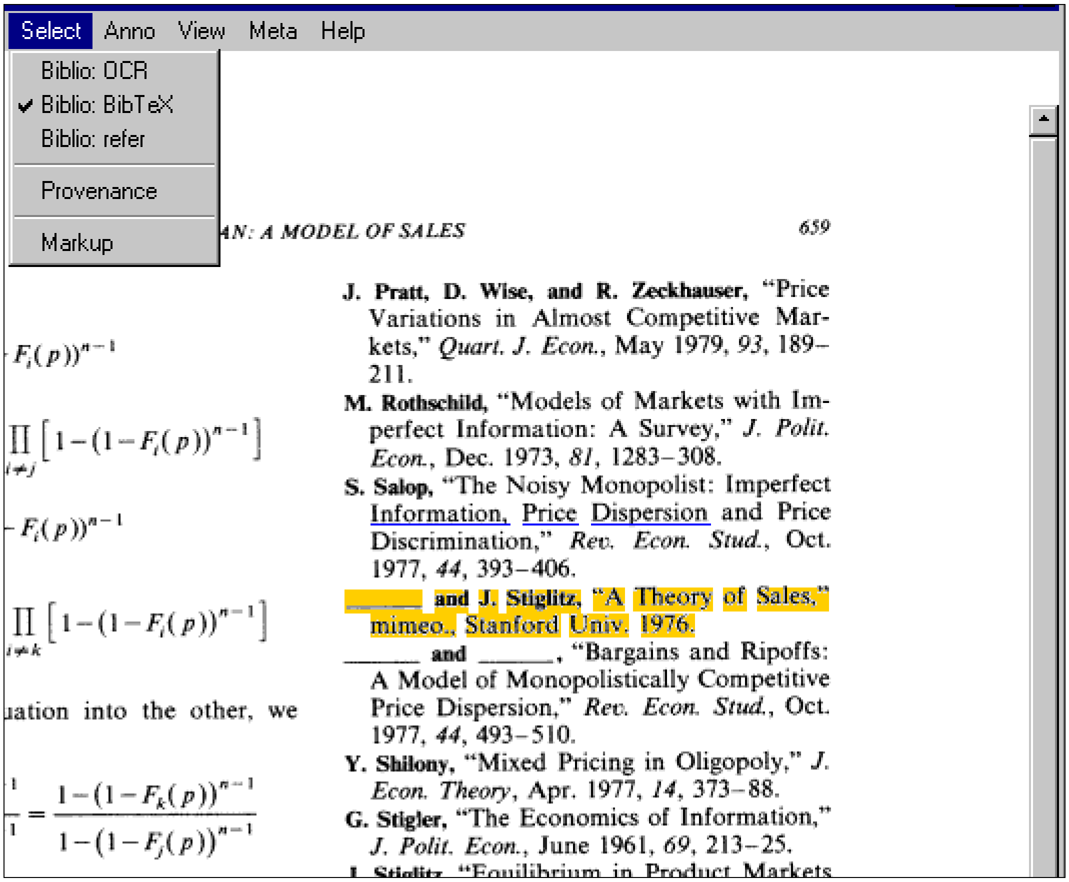
Lenses are a good example of a kind of annotation that is not readily available via paper. First, lenses can perform functions that are not easy to do with paper, such as magnifying a portion of a document, or containing an active link to another part of a document.

Other examples of lenses that are primarily annotative are under development. For example, one such lens shows information about Chinese characters in Chinese text, to be used as a teaching aid. Another shows language translations, from a layer of aligned language translations.

5.1.3 Structural behaviors

Structural behaviors hook into the IDEG, representing a function applicable to a structurally meaningful portion of the document. Whenever an action is happening in that area of the document, structural behaviors are given an opportunity to modify the results. Structural behaviors can invest incremental knowledge into a document or leverage existing structure.

As an example of a structural behavior that invests some incremental information, recall that the user can select words in the document image and paste the corresponding OCR. If further structuring can be imputed to a region, it may be useful to paste different text more directly suited to another application's input. In Figure 14(a), a bibliographic entry has been selected. To incorporate this entry into another application, one could start by pasting the OCR text and editing it as necessary. Instead, we have created a "Biblio" behavior that automatically performs some useful transformations. Specifically, having a semantic description of fields for author, title, pages and so on, the Biblio behavior affects the selection protocol, automatically inserting BibTeX- or



(a)

```
______ and J. Stiglitz, ``A Theory of Sales,'' mimeo.,
 Stanford Univ. 1976.

@Article{
        Title = "Theory of Sales",
        Author = "S. Salop and J. Stiglitz",
        Journal = "mimeo., Stanford Univ",
        Year = 1976
}

%T Theory of Sales
%A S. Salop
%A J. Stiglitz
%J mimeo., Stanford Univ
%D 1976
```

(b)

Figure 14: An example of structural behaviors. (a) shows a page image in which behaviors associate with the subtrees corresponding to bibliographic entries the semantic contents of those entries. Here the user has selected the entirety of one entry, and then uses the entries under the "Selection" menu to change the way selection works so that content of the bibliographic entry is formatted as requested before being put in the selection buffer. (b) shows the results of choosing the "OCR", "BibTeX", and "refer" menu entries, respectively, from this menu, prior to selection, and then pasting the resulting selection in another application.

refer-formatted text, as the user chooses. Once in the selection buffer, of course, this formatted text could be pasted into any application, as evidenced in Figure 14(b), which contains various transformations of the text pasted into another application. The formatted text is computed on the fly, so that adding an additional output format merely requires coding the appropriate formatting statements.

A similar "alternative select and paste" has also been implemented for mathematics, with a fixed set of output formats (Lisp and TeX) available at this time.

This example of structural annotations is not intrinsically annotative. From the point of view of a user, the ability to select bibliographic alternatives can simply be a useful document feature. In this example, the alternatives were added by someone other than the original author, lending them an annotative quality. Some more obviously annotative examples are described in the next section.

5.1.4 Combining Annotations

It is possible to combine behaviors together in useful ways. In some cases, it is merely convenient to use a number of forms of annotation together to a single purpose. For example, one might attach a note to describe the overall comments, adding a hyperlink to move the reader to an annotated portion of the text, where additional annotations appear, e.g., as yellow highlights or blue-pencilled markup.

Other examples more closely couple multiple annotations to form coherent annotation genres. One example with which we have experimented is a "language translation" lens. This lens makes use of structural annotations that build sentence-aligned translations from additional layers. (In our case, these aligned translations were constructed manually, although the model is indifferent to their origin.) When positioned over a line of a sentence, the translation lens displays underneath that line the corresponding translations, in our case French or German. (One could just as easily create a "biblio" lens that displays variously formatted text, and a translation selection behavior that allows the pasting of alternative translations.)

Another, more elaborate example, can be thought of as simulating creative paper folding. Paper documents can be laid out in space to gain a sense of overall structure and to cross-reference specific information across pages [OS97]. An orchestrated combination of multivalent annotations can exploit computer processing of the document to provide a good portion of this useful functionality.

Notemarks, a fusion of "note" and "bookmark", is a form of annotation that provides some of the same ability to sense the overall structure of a document by combining several kinds of annotations together. Notemarks were first prototyped using the Tk toolkit's text widget [Oust94] and applied to the visualization of UNIX online documentation in TkMan [Phel94]. The same functionality has been duplicated in MVD, via the combination of several types of behaviors. First, structural annotations are used to allow a user to collapse or expanded a section by clicking the section header. Within otherwise collapsed outline sections, single lines are made visible by versions of span annotations similar to those described above, but in which care was taken that the graphics properties of the span annotations were higher priority than that of structural annotations, and thus can override the visibility state. Some such spans are created just for the purpose of overriding a structural collapse. For example, often one refers to a manual page just to check the letter of a command line option, so it is reasonable to pre-configure these lines as visible within collapsed sections. Similarly, the first line of each paragraph of commonly important sections can be excerpted in order to present a highly informative single screen overview. In addition, lines in which the individual user may have somehow indicated an interest might also override collapse.

To implement Notemarks, an MVD media adapter was written especially for ASCII UNIX manual pages. This media adapter examines an ASCII layer looking for manual page structures, such as section headers and examples, and creates an IDEG with spans and attached structural behaviors that present the page in a stylized fashion. Figure 15 illustrates an MVD document so created from the text of the "file" command manual page. The headings of each section are shown in large font, preceded by a triangle whose orientation shows the state of section collapse. Initially, the sections are all collapsed, so their contents are generally not visible. However, command line options have spans that override the collapse, so they are visible. (These appear in the "Description" section). Also, as mentioned above, lines in which the user expresses an interest should be shown. In this example, lines containing search matches (for "Windows" and "Mac") also override the collapse of sections. Clicking on any of these lines opens the document to the desired point.

Figure 15 is also noteworthy because it illustrates MVD operating on another document format, namely, plain text. (Actually, the format supported here is plain text that conforms to a UNIX man page; support for general plain text is also provided, but by a separate, simpler media adapter.) Getting MVD to support this format was simply a matter of writing a behavior to support it, i.e., a media adapter that builds the IDEG from a plain text, lays out the tree on the canvas appropriately, and so forth. All the behaviors described above are immediately available.

The Notemarks idea works well together with other annotations. For example, the user can (using a menu entry) determine whether highlights or copyediting marks should be treated like Notemarks, and hence override section collapsing. Thus. turning on the Notemarks in the HTML collapses the document long major HTML structures; specifying that copyediting marks should override this folding makes most other text invisible. Figure 16 shows the document marked up as in Figure 8, but collapsed so that most of what is visible is lines with copyediting marks.

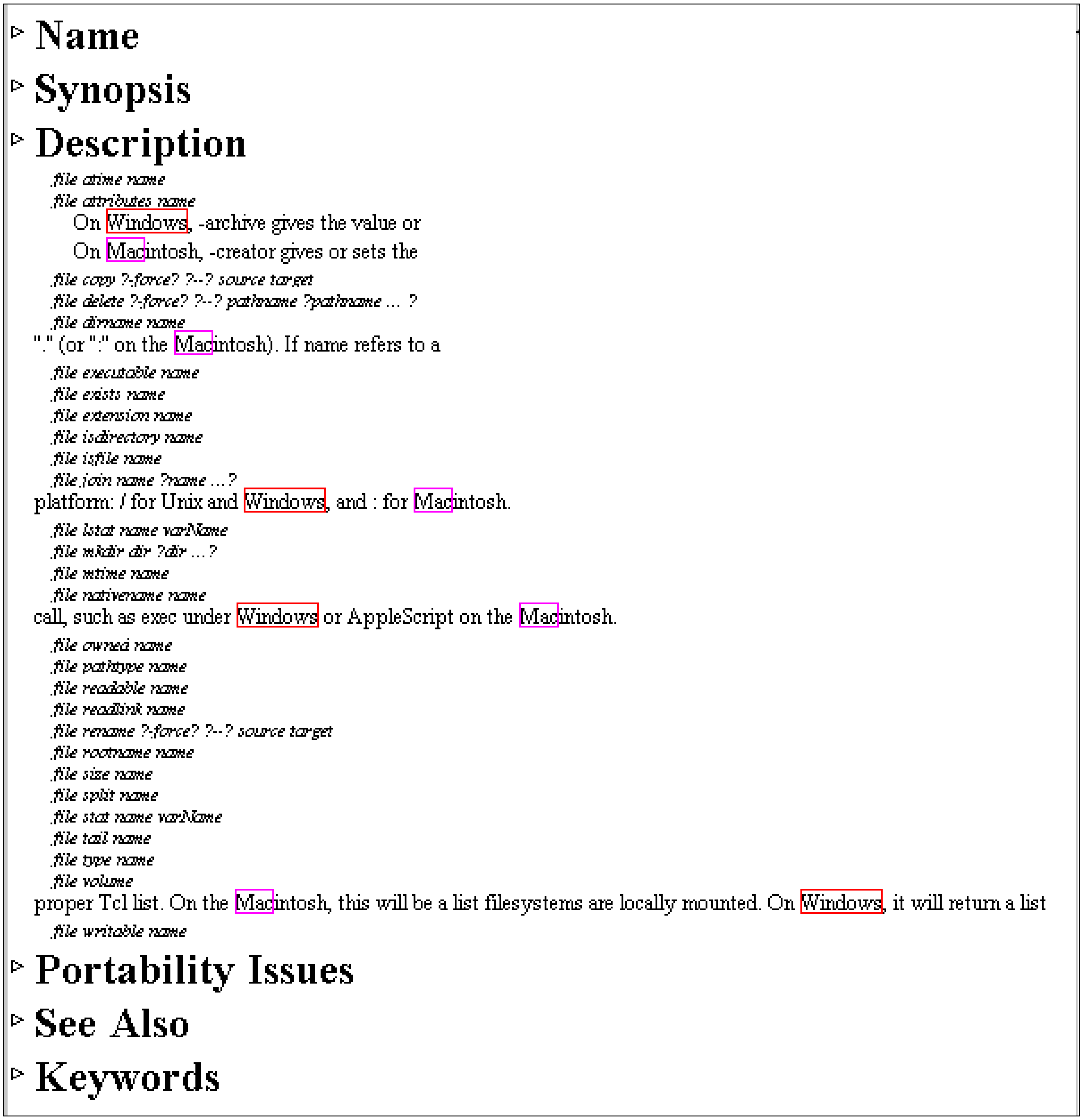


Figure 15: Notemarks: Within otherwise collapsed Description section, subcommands, highlights and search hits show through. Clicking on the desired area opens it up to that point.

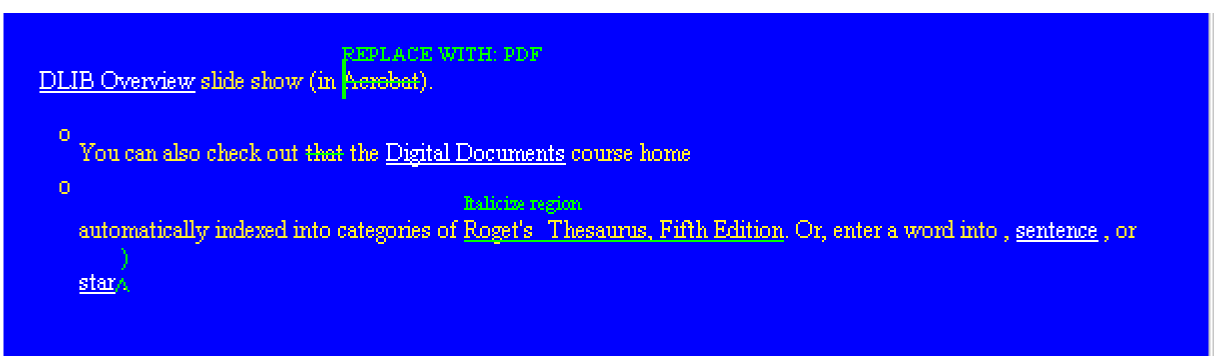


Figure 16: Copyediting marks used as Notemarks

6 Applications to Other Data Types

So far, we have examined documents built from distributed layers, but whose primary content is rather standard, i.e., text and scanned images. We have made some attempts to extend the model to other data types. We have begun to experiment with data that have temporal extent, data that have spatial extent, and data that have both temporal and spatial extent. Our work in this area must be regarded as preliminary, but our experiments thus far indicate some possibilities and limitations.

6.1 Data with Temporal Extent

Data with temporal extent includes sound and video. Video and sound have been incorporated both as separate elements, and as data associated with a structural annotation. However, at this point, the internal structure of these media types is largely opaque to the infrastructure.

By separate elements, we mean that sound or video can be the content of a note. When such a note is displayed, controls become visible when the cursor enters the boundary of the note, and the content is streamed. It is also possible to use video or sound as annotations. In this case, the user can select a span, and choose a behavior for making video or sound annotations. A widget is displayed requesting a URL. A tool to control the continuous medium is displayed to the right in the margin near the selected text.

It is possible to associate textual notes with video notes, and synchronize the text shown with the video, to achieve a captioned video effect. Figure 17 shows an example of a document with a sound annotation, and a video note synchronized with two textual notes, one in English and one in Polish. This example illustrates a potentially useful application of the document model to continuous media, in that keeping the data layers separate in the document, with the ability of combining them in the client, leads to flexible captioning.

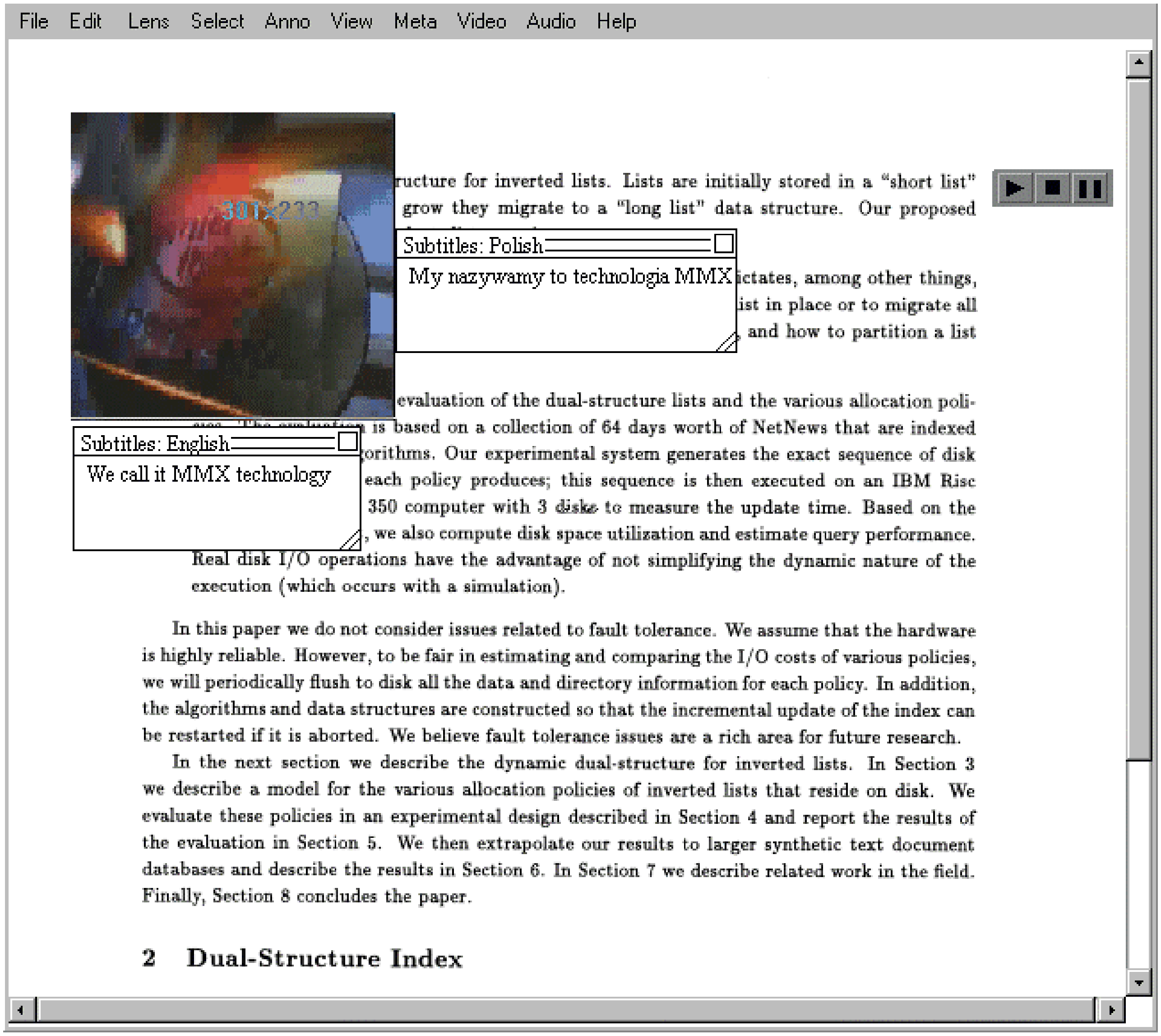


Figure 17: Data with temporal extent: The widget on the right is an anchored sound annotation. The floating window is a video note. The two textual notes are captions synchronized with the video stream.

6.2 Data with Spatial Extent

We have experimented with applying the idea of MVD to visualizing and annotating data with geographic interpretation, i.e., maps and the like. However, as we have not yet attempted to integrate this effort into the MVD framework proper, we forego discussion of this experiment here. The interested reader can examine our prototype, called the GIS Viewer, which is available as another Java applet at http://elib.cs.berkeley.edu. (The "GIS Viewer tour" is recommended.)

7 Using MVD for Annotation

The previous sections provided an overview of MVD, and gave examples of annotative behaviors. Here we clarify some pragmatic details of the processes by which a user might use MVD annotatively.

To begin with, a user opens a multivalent (i.e., hub) document in the MVD applet. This may be done by specifying the URL for a hub document to the MVD Open widget, available under the File menu. Since the MVD applet may not already be running, we have provided a service, using a CGI script, to which a hub document URL may be provided as an argument. The script returns a call to the MVD applet, passing it the hub document URL as an argument. In other words, by appending a hub document URL as an argument to this script URL, one produces a valid web URL which, when supplied to a Java-compliant web-browser, will invoke MVD on a given hub document.

Once the hub document is loaded, the user can use whatever behaviors there are that have been specified by that hub. Presumably, these will include the various annotative behaviors described here, although exactly what is included is specified by the hub document. The user makes various annotations on the document. (As explained above, from the MVD viewpoint, all the user is doing is creating new instances of MVD behaviors; these just happen to be annotative.) Then the user saves these annotations by created a new hub document. This is done simply by selecting the “Save As” entry in the File menu, and supplying a name. MVD will create a new hub document, which will include the instances of behaviors extant at the time, including the new annotations.

Figure 18 contains the simple hub document resulting from annotating a web page with a replacement copyediting span and a note, and then saving, as just described.

```
<MULTIVALENT GENRE="HTML"  SEARCHNB="ON"  ANNONB="ON" COPYEDNB="ON">
<Multivalent.std.adaptor.HTML  BEHAVIOR="Multivalent.std.adaptor.HTML"
URL="http://www.darpa.mil/"></Multivalent.std.adaptor.HTML>
<Layer  BEHAVIOR="Multivalent.Layer"  NAME="Personal"  URL="inline">
<Span  BEHAVIOR="Multivalent.std.span.ReplaceWithSpan"  CREATEDAT="Wed Feb 11 14:52:02 PST
1998" NB="COPYEDNB"  INSERT="DARPA">
<Start  BEHAVIOR="Multivalent.Location" TREE="0  20/It 4/P 0/BODY 0/HTML"  CONTEXT="It
(DoD). manages"></Start>
<End  BEHAVIOR="Multivalent.Location"
          TREE="2  20/It 4/P 0/BODY 0/HTML"  CONTEXT="It (DoD). manages"></End>
</Span>
<Note  BEHAVIOR="Multivalent.Note"  NAME="NOTE1739298548"  X="161"  Y="258"  WIDTH="287"
HEIGHT="303"  POSTED> Here are some comments on \nthe DARPA home page. \nRW</Note>
</Layer>
</MULTIVALENT>
```

Figure 18: A hub document resulting from annotating a web page with a span and a note. The base document is the DARPA home page. As this is an HTML item, the appropriate media adapter is specified. The ReplaceWithSpan instance makes the editorial suggestion that an particular occurrence of the word “It” be replaced with the term “DARPA”. The start and end portions of the span show tree locations and redundant context for the location.

Since MVD is an applet, it cannot save to arbitrary locations. To get around this limitation, all saves are done to a scratch directory on our server to which the world has read/write permission. While items in this scratch area are apt to be transient, the user can subsequently visit the hub documents via a web browser, to which they will merely be ASCII files. The web browser can then be asked to save the contents on the user's web server or other networked service at which it will have a valid URL. The hub documents contain only absolute references, so the resulting item is a persistent well-formed MVD hub document.

There are several other augmentations we have provided to facilitate use of MVD. While strictly speaking MVD can only open hub documents, one can specify a default hub document for given data types. We have done so, and extended MVD's Open procedure so that, if it is given a recognized MIME type instead of a hub document, it will create a default hub document for this data type and open that. In this manner, one can use MVD as a web browser: One can specify to Open an HTML page rather than a hub document, and MVD will open a synthesized default hub document containing this page as a layer. If the HTML page contains hyperlinks, these can be selected for traversal within MVD, which will once again open the web page by inferring a default hub document around it.

Another mode in which one may use MVD is email. An MVD emailer has been written by Hoon Kang. This allows one to read, originate, and reply to one's mail within MVD. The user may respond to an ordinary email message by using MVD annotations, and mail off the result. Similarly, the user may mark up a document, and email rather than save the hub document. The email contains both the hub document and a URL for opening a temporarily stored version of the hub document. If the recipient is using MVD to read mail, the hub document can be viewed directly; in other mail reading programs, the user can invoke the URL, thus running MVD on the hub. We have found MVD to be useful in this manner for replying to long ASCII email messages.

7.1 Evaluating Multivalent Annotations

Multivalent annotations meet many of our requirements stipulated for annotations simply by virtue of operating within the multivalent document model. They are *highly expressive* (behaviors have access to every state of the fundamental document life cycle), *extensible yet composable* (new types of annotation can be created by writing new behaviors; the multivalent framework insures composability of behaviors conforming to the protocols), *distributed and open* (layers and behaviors are intrinsically distributed; one can annotate any document for which one has a media adapter, with no server-side requirements), *format-independent* (behaviors manipulate the abstract document tree and communicate to encapsulated media types through media adapters), *platform-independent* (the infrastructure is written in Java).

Another desideratum of digital annotations is that they appear *in situ*. We meet this requirement by having the individual annotative behaviors rely upon the geometric placement information of behavior instances available in the *format* stage of the document life cycle. Annotations are attached to a particular component or series of components, and then placed in relation to them. Thus when a document is double-spaced, say, or a table sorted, the annotation is drawn at the right place because it is drawn in relation to the new position. Placement is managed by the multivalent framework calling a behavior method at the right time.

In addition to meeting these conditions, multivalent annotations also deliver at least some of the promise of digital annotations. They can be readily shared with others, and can have dynamic properties, such as being executable or dynamically changing the view of the document. Exploiting other aspects of digital annotations await future work.

8 Future Work

8.1 Other Data Formats

We have shown that MVD can be applied to multiple document formats, namely, scanned images, plain ASCII text, and HTML. However, other document formats are in widespread use, including those used by common office suites, i.e., word processing, presentation, and spreadsheet formats. Support for "near image" formats, i.e., PostScript and Acrobat, would also be valuable. One possibility is simple to provide media adapters for each media type. Such behaviors could run efficiently, but requires significant programming. Another possibility is to write a media adapter for RTF, and provide a service that converts documents in and out of this format. Another possibility involves support for XML. We plan to support XML along with some style sheet language. (Indeed, our support for HTML was implemented with this in mind, so support for XML should not be a large step.) Of course, support for XML may be useful in its own right. In addition, it should possible to translate most proprietary document formats into an XML DTD plus style sheet. Thus, given support for XML, one needs only a translation service to convert the given proprietary format into XML. Perhaps the office suite producers will include export and import of XML, which will save us the trouble of providing translation software or services.

Our coverage of HTML is not complete. Some aspects are straightforward, and just require more programming on our part. Others are less obvious. For example, it is not clear to us what is best to do with dynamic components in documents, just as JavaScript or Java.

Near image formats can be handled in a number of ways. One can render documents by converting them to image formats; alternatively, one can wait for promised 2D graphics rendering capabilities to appear in the Java class libraries. The underlying document representation could be used to more easily compute structure (i.e., word location). (This is straightforward for PDF; may packages are available to do so for PostScript, with varying decrees of accuracy.) Alternatively, it is possible just to run OCR on the images. Doing so produces an accurate rendering of the text content of the documents, especially when true images containing text are included in the document, although at greater computational cost. (We have set up a simple service into which the user supplies a URL for a tiff file produced by a scanner; the service runs OCR on the contents, converts the image to GIF for viewing, and returns a MVD document combining this image and OCR contents. A similar service could be set up to other types of conversion.)

We have shown other applications that apply the ideas of MVD to other data types, namely, those with temporal and spatial extent. We have demonstrated some degree of support for video and audio within the model proper. It remains to be seen whether these data types can be well integrated into a single framework.

8.2 Authoring Behaviors

We have provided some rudimentary authoring behaviors, namely, those for authoring notes and various kinds of spans. However, more work is required in this area. In particular, the following features are desirable:

- The ability to edit or remove annotations once they are created.
- More complex kinds of copyediting annotations. For example, one would like to be able to indicate that a span of text should be moved to another point. Doing some presumably requires rendering graphics, possibly across pages.
- The ability to provide recursive annotations. Right now, one can, e.g., put copyeditor marks or hyperlinks in a note, but one cannot put a copyeditor span on another copyeditor span.
- The ability to save edited base layers. Right now, one can execute a copyeditor mark, or otherwise modify a layer, but not of the media adapters currently write out their contents upon a save. Doing so is just a matter of programming.

8.3 Other User Interfaces

Furthering our goal of "paper as a user interface", we would like to capture annotations while writing upon paper rather than using a keyboard and screen. For example, the user might mark up paper copy, and provide this as input to a scanner. Then the original document could be subtracted to produce a mark-up layer, which would then be subject to recognition analysis. The resulting layer could be executed interactively by a user, in effect automatically executing one's paper mark-up. Alternatively, the marks might be made on paper but captured by using a special pen or a video camera.

8.4 Dynamically Assembling Document Components

With layers and behaviors generally, but especially with annotations, one would like to take a document and collect commentaries on it (i.e., retrieve all annotations by various authors). We believe MVD hub documents can be searched in this fashion using generic Web search engines. Within MVD, one would like to support collections of annotations, so that one can flip back and forth between the annotations of different commentators on the same document. Similarly, one would like to be able to annotate already annotated documents simply by including a hub document inside a hub document. This idea of "cascading hub documents" should be straightforward to support.

8.5 Real-time collaborative authoring

We have mostly described an asynchronous view of collaboration, in which documents are visible after they have been authored. Indeed, our "document-centric" view of collaboration is distinguished from session-oriented collaboration models in this way. However, it is possible to allow synchronous collaboration, in our model, which amounts to allowing one user to see a layer as it is being authored by another. Hence, we plan to explore extending the system to support real-time viewing of layers as they are being authored. One possible way to implement this functionality in MVD is via NCSA's Habanero[3], a session-oriented object-sharing framework, written in Java, which enables software developers to transform single-user applications into multi-user, shared applications.

9 Related Work

There has been a great deal of prior work in the area of document models and computer-based collaborative work that we have drawn upon in this research. We do not attempt to review this body of work here, but instead, compare and contrast our work to some of these approaches, with the goal of highlighting how we see our work as differentiated from them.

It is useful to first contrast the MVD model with more traditional document models. Most document processing tools manipulate documents of a specific data type. E.g., Microsoft Word handles documents in native Word format or RTF; Web clients handle HTML; emacs processes text; Adobe Acrobat[4] products handle PDF. For the most part, these tools also generally have rather limited notions of extensibility, and of a relatively coarse-grained level. Thus, it might be possible for users to add a macro to Word, but it is very difficult to add a new feature. Moreover, customizations are local to users or installations, and hence, difficult to share. Some systems, notably emacs, address this problem by providing a powerful extension language. This approach has been successful in some ways. But one is still left with the intrinsic limitations of the original framework, which have proven difficult to overcome, as well as an extensive, support-intensive, code distribution problem.

The MVD model addresses these problems in a number of ways: First, the incorporation of layers as a basic document structuring element allows for a modularization of functionality, in that not every piece of functionality need be supported by a single mark-up language. For example, to incorporate hyperlinks into Word, say, the designers must support hyperlinks in the underlying mark-up language, and then modify the software to handle them (and resell the new version). However, in the MVD model, this functionality could be added to a Word document without the underlying Word machinery even being aware of it. Moreover, the same hyperlink mechanism would apply to other mark-up languages.

OpenDoc [Appl95] and OLE/COM [Broc95] represent document structuring efforts that address some of the same issues as MVD. OLE is Microsoft's propriety approach; OpenDoc a cross-platform approach supported by a consortium (until its recent demise). OpenDoc and OLE are similar to each other in that both view documents as comprising multiple embedded document segments, each to be interpreted by software components. Both are similar to MVD in that they are all essentially software component technology, and they attempt to simplify document structuring

[3] http://www.ncsa/uiuc.edu/SDG/Software/Habanero/index.html
[4] http://www.adobe.com/prodindex/acrobat/main.html

via modularity. However, OpenDoc and OLE attempt to modularize by dividing a document into planar regions, each of which is controlled by a separate process. E.g., a table embedded inside a text document would be managed by a table editor module, while the containing text component is managed by a text editor module. MVD in effect provides a third dimension. That is, an OpenDoc or OLE document would simply be one layer of an MVD document, albeit a modularized one.

Note that additional modularity is provided by MVD’s layering-and-behaviors approach: With only the OpenDoc/OLE model, enhancing functionality requires changing individual software components. While there is some mechanism for inter-application communication, it not generally possible to augment a part, e.g., to add a hyperlink or a lens to a component that does not already support such functionality.. Rather, to add, say, hyperlink support to both tables and text, both the table and text components would have to be modified. In MVD, hyperlinks can be added via a separate layer, i.e., in a “depth” dimension, without having to modify *either* an underlying text or table manager.

OpenDoc/OLE components are essentially black boxes. In contrast, MVD integrates information from multiple layers into a single document representation. Thus, in MVD, it is possible to introduce behaviors that operate on multiple data types, whereas in OpenDoc/OLE the individual editors are entirely separate programs. Thus, it is fairly straightforward to introduce behaviors like a magnifying lens into MVD, and have it operate over multiple formats, whereas it is not obvious how to implement an analogous function in OpenDoc or OLE.

OpenDoc and OLE require machine-specific implementations of software components. Thus, even though an OpenDoc document is in principle cross-platform, there is no guarantee that a required OpenDoc editor will be available on a given platform, much less installed. MVD addresses this problem by the use of Java, so that code is transferable and normally down-loaded automatically across the network.

Finally, OpenDoc and OLE both embrace a model in which a user controls a document, which resides as a file in a single location. As such, these models are not particular conducive to the sort of collaborative possibilities easily introduced in MVD.

Now let us contrast how we have used MVD to support annotations with the many other models of annotation. One set of systems consider annotations to be part of the document per se, and therefore, require write-access to or copying of a document in order to annotate it. For example, Adobe Acrobat and Microsoft Word allow several different kinds of annotations. These may be added after the document is authored, but only by editing a write-able version of the document. Hence, they are ill-suited for distributed, loosely coupled collaboration.

These systems are also entirely data-type specific. E.g., Acrobat allows the annotation of PDF files, but only PDF files. Similarly, Word’s annotations will not help a user who has authored a file using some other word processor (much less one who has only a scanned image to work with). In contrast, the MVD model will allow annotation of any data format that is supported. In general, MVD is capable of incorporating diverse document formats, and operating on them.

Other models tend to support only limited and very specific forms of annotations, e.g., notes, or hyperlinks. MVD already supports forms of annotation not readily available elsewhere, e.g., executable copy-editing. Moreover, in MVD, there is no limit to the kinds of annotations one can envision and implement. As end-users can introduce new behaviors in their documents, new forms of annotation and collaboration can be introduced dynamically.

Other models of annotation presume closed systems. I.e., they require that a user be registered in some fashion in order to participate in collaboration. In contrast, MVD is highly open, meaning that one need only be able to read a document in order to annotate it; one need only be able to write to some networked resource to be able to share one's annotations. The original document author need never even know that the document has been annotated, any more than one needs to know that a hyperlink points to one's web page.

There are a number of existing systems that support various kinds of in-place annotation. These include the annotations facility in Lotus Notes™, which requires making available "hooks" for annotation attachment in a given document. ForComment ™ supports individuals in a group making comments on documents in most common word processing formats. Markup™ [5] supports annotation, including copyeditor marks, in the MacIntosh environment; the NeXT OS provides blue-pencil mark-up over any document rendered as Display PostScript. These operate at the graphics level, and hence have the nice property that any document can be annotated. However, annotations are superficial. (I.e., one can't attach a copyeditor mark to a span, say, and then execute it.) All these models require buy-in to a particular system.

Adobe's Acrobat bears a number of similarities to the scanned page image application of the general multivalent model. Adobe has published the specification of the PDF format viewed by Acrobat, and that format is in principle extensible by anyone. In practice, however, it is extensible only by Adobe as extensions to the format require corresponding changes to the viewer, and that is proprietary to Adobe, and would be difficult to build. Moreover, the types of annotations provided in Acrobat, though growing with each new version, are geometrically positioned at x,y coordinates on the page, not tied to content, and therefore not robust to changes in the document.

Microcosm [FHHD90] builds a hypertext system on top of existing applications, on Microsoft Windows. The philosophy is that, rather than compete with Microsoft Word and Excel, one should take advantage of the hooks these applications make available and layer the system on top of them. While gaining the power of highly evolved applications, this strategy is limited to the extent that such hooks are permitted by those applications.

ComMentor [RMW95] focuses on the server side of annotation support. ComMentor has a complete and well worked out meta data strategy and a database system for managing annotations, but only provided minimal functionality at the client. The ComMentor server-side support would be a nice complement to multivalent annotations.

Knowledge Weasel [LS93] distinguishes between surface annotation and deep annotation. Like ComMentor it focused on database aspects like a common record format and surface annotations and not as much on deep annotations, except for spatial data imagery. It took advantage of common

[5] http://www.mstay.com/

tools (Tcl and Tk) for wide availability, but ultimately it was also limited to them. If the text widget does not admit, say, lenses, then that is an insurmountable barrier to implementing them.

Microcosm, ComMentor and Knowledge Weasel were developed mindful of the fact that it takes a great deal of effort to build a document formatter-renderer, and hence follow a strategy of interoperating with existing formatter-renders. Unfortunately, requirements of deep and powerful annotation push against the limits of such systems. Instead, we pursued a strategy that imposes an up-front cost to bridge existing application formats into the model and to reproduce the desired pieces of functionality. Our hope is that this price will be ultimately be worth paying.

10 Conclusions

We have outlined the MVD architecture, and demonstrated its use in enlivening legacy documents and supporting distributed annotation. We have demonstrated the model's applicability to multiple document formats, and suggested how it might apply to other data types.

There are a number of features of our approach that one might view as drawbacks, or cause for concern. One particular important concern is the choice of Java as the implementation language. Java provides a number of advantages: portability, sizeable and growing support in the form of tools, compilers, program development kits, near-platform independence. However, Java currently has a number of shortcomings: Implementations are immature; the Java VM/browser interface is often defective, and do not utilize memory well. Moreover, there has yet to emerge a stable model, much less decent implementation, of how Java programs running within browsers change persistent state. The last issue is of serious concern to us, as a substantial part of our effort is directed at authoring. (Indeed, the problems are significant enough for us to consider abandoning MVD as an applet, and running it as a stand-alone Java application.) The language definition is also continually changing. We note that these problems are generic to Java, and there is substantial motivation for industry to address them. Of course, there is nothing in our model that requires it to be implemented in Java, or, for that matter, for the extension language (i.e., the language in which behaviors are written) to be the same as the implementation language.

However, the model does require behaviors written in a single language. Thus, we cannot easily take advantage of the great amount of related code that is written in C, C++ or the like (or, if we do so, we lose some platform independence, and transparent installation). This problem seems to us to be an intrinsic trade-off between platform independence and using code targeted to a particular architecture, rather than a defect specific to MVD.

We suspect that some of the individual MVD applications we envision could be realized more efficiently as stand-alone, platform-specific applications. Our claim is that there are large benefits to having a ubiquitously available infrastructure that facilitates extensible documents, both as a practical matter and as a platform for research. The ease with which we were able to create the annotation types and the assorted "power tools" described above illustrates this point. We are relying on continuing improvements to just-in-time compilers and faster processors to address performance concerns. However, occasionally, we have found ourselves adding some more support to the infrastructure where efficiency has become an issue.

While MVD is still a research prototype, our experience with it indicates that MVD is a useful step toward a more network-centric document model. In particular, we have shown that it can run on multiple platforms, handle multiple document formats, provide a high degree of extensibility, and provide a basis for “spontaneous collaboration”. Perhaps multivalent documents do not make digital documents as inviting as paper, but they move the digital medium one step closer.

11 Acknowledgments

Many members of the UC Berkeley Digital Library Project contributed to the ideas in this paper. We are especially indebted to Gary Kopec for his insights into the use of scanned document images, and for helping us untangle the undocumented intricacies of the XDOC format.

Loretta Willis implemented the table sorting behavior in the current MVD implementation. Wojciech Matusik implemented the video and audio annotations. Hoon Kang implemented services to regularize HTML, and also implemented the MVD emailer.

Other Digital Library Project members, including Ginger Ogle and Joyce Gross contributed to the many discussions from which MVD arose. The alternative selections for mathematics derive from the work of Richard Fateman and Taku Tokuyasu on optical character recognition and representation of mathematics.

We would like to thank Carl Staelin, Nic Lyons, Ben Vigoda and Steven Rosenberg of Hewlett-Packard Laboratories for their encouragement, support and their many valuable suggestions.

12 References

[Appl95] Apple Computer. OpenDoc Programmer’s Guide. Addison-Wesley, 1995.

[Bier93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and Magic Lenses: The see-through interface. In Proceedings of SIGGRAPH `93, pages 73-80, August 1993.

[BS97] Tim Bray and C. M. Sperberg-McQueen. Extensible Markup Language, Working Draft. http://www.w3.org/TR/WD-xml-lang

[Broc95] Kraig Brockschmidt. Inside OLE 2. Microsoft Press, 1995.

[DD94] DeRose, Steven J. and Durand, David G., Making Hypermedia Work: A User's Guide to HyTime. Kluwer Academic Publishers, 1994.

[FHHD90]A. Fountain, W. Hall, I. Heath and H. David. Microcosm: An Open Model for Hypermedia with Dynamic Linking. In Proceedings of ECHT `90, 1990.

[Hal88] Frank G. Halasz. Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems. Communications of the Association for Computing Machinery, July 1998, pages 836-852.

[Hal91] Frank G. Halasz. Seven Issues Revisited, Keynote Address, Hypertext '91 Conference San Antonio, Texas December 18, 1991, http://www.parc.xerox.com/spl/projects/halasz-keynote/

[Lad] Richard Ladner. Sticky Pointers. University of Washington Technical Report.

[LS93] Daryl T. Lawton and Ian E. Smith. The Knowledge Weasel Hypermedia Annotation System. In Hypertext '93 Proceedings, November 14-18, 1993, pages 106-117.

[LM95] David M. Levy and Catherine C. Marshall. Going Digital: A Look at Assumptions Underlying Digital Libraries. Communications of the Association for Computing Machinery, April 1995, pages 77-84.

[Mars97] Catherine C. Marshall. Annotation: From Paper Books to the Digital Library. Proceedings of the Second ACM Conference on Digital Libraries, July 23-26, 1997.

[Oust94] John K. Ousterhout. Tcl and the Tk Toolkit. Addison-Wesley, 1994.

[OS97] Kenton O'Hara and Abigail Sellen. A Comparison of Reading Paper and On-Line Documents. In Proceedings of CHI '97, March 22-27, 1997.

[Phel94] Thomas A. Phelps. TkMan: A Man Born Again. The X Resource, 10, 1994.

[Phel98] Composing Multivalent Documents. UC Berkeley Ph. D. thesis. In preparation.

[PW96a] Thomas A. Phelps and Robert Wilensky. Multivalent Documents: Inducing Structure and Behavior in Online Digital Documents. Proceedings of the 29th Hawaii International Conference on System Sciences, January 3-6, 1996.

[PW96b] Thomas A. Phelps and Robert Wilensky. Multivalent Documents: Architecture and Applications. In Proceedings of the First ACM International Conference on Digital Libraries, March 20-23, 1996, pages 100-108.

[PW97] Thomas A. Phelps and Robert Wilensky. The Architecture of Multivalent Documents. In preparation.

[RMW95]Martin Roscheisen, Christian Mogensen and Terry Winograd. Beyond Browsing: Shared Comments, SOAPs, Trails, and On-line Communities. In Proceedings of the Third World Wide Web Conference, April 10-14, 1995.

[SH97] Abigail Sellen and Richard Harper. Paper as an Analytic Resource for the Design of New Technologies. In *Proceedings of CHI '97*, March 22-27, 1997.

[Trig83] Randall H. Trigg. A Network-Based Approach to Text Handling for the Online Scientific Community. Ph.D. Thesis, Dept. of Computer Science, University of Maryland (University Microfilms #8429934), November, 1983.