Multivalent Annotations[1]

Thomas A. Phelps and Robert Wilensky
University of California, Berkeley
{phelps, wilensky}@CS.Berkeley.EDU

Abstract

Paper is still preferred to digital document systems for tasks involving annotating, folding, juxtaposing, or otherwise treating the document as a tactile object. Based on the Multivalent Documents model, Multivalent annotations bring to digital documents of potentially any source format, from PostScript to SGML, an open ended variety of user-extensible, sharable manipulations. Several very different forms of distributed annotation based on this model have been implemented. The Multivalent framework composes together annotations of any type, which can result in novel, useful combinations.

1. Introduction

Digital documents are superior in many ways to their paper counterparts. They are easier to edit, reproduce, distribute, and search than paper documents. While these advantages have led to the dominance of the digital format for document preparation, paper still provides much functionality that is simply unavailable via the digital medium. An important class of capabilities that exploit the special affordances of paper is annotation. By annotation, we include a large variety of creative manipulations by which the otherwise passive reader becomes actively engaged in a document. 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.

Having annotations in digital form would immediate confer upon them the many benefits unadorned documents already enjoy from digitalization. In addition, the digital format would provide the possibility of entirely new forms of annotation, as it allows for the possibility of dynamic annotations, in addition to more conventional passive commentary.

If digital annotations hold great promise, several practical matters must be addressed before they become an fundamental part of the work environment. We suggest that for digital annotations to succeed, they must possess the following properties:

Appearance in situ. Annotations should appear *in situ*, that is, on the documents themselves. 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.

1. This research was funded as part of the NSF/NASA/DARPA Digital Library Initiative, under National Science grant number IRI-9411334.

Published in Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries, 1-3 September 1997, Pisa, Italy.

Highly expressive. Annotation 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 copyeditor's detailed corrections. In addition, annotations must be able to provide arbitrarily powerful forms of user interaction, providing active capabilities in addition to passive markings.

Format independent. Annotation systems must work with a variety of source document formats. Users should be able to continue to use their preferred document preparation systems, yet produce documents amenable to annotation.

Extensible, yet composable. 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 (that is, new technology) are devised, the new should be seamlessly integrable with the old. That is, 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. Anyone should be able to annotate any document they can view. Hence, annotating must require no special privileges. Thus, while annotations need to appear as if part of 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. No customization of the original document's server should be required for a given individual to be able to make an annotation available. Annotation must be a completely open process, with the resulting annotated document an object existing as a set of distributed network services.

Platform independent. Since annotated documents need to be 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.

Robust. As an annotation may reside 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.

While many forms of annotation capability 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. In particular, as we describe further below, 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 have implemented a set of annotation capabilities that satisfy most of the requirements above. We call our proposed solution *Multivalent annotations*. Multivalent annotations appear *in situ*, are mutually composable, source format independent, extensible, seamlessly integrable, immediately portable over the network, server independent and robustly positioned. We have implemented our annotations using our Multivalent Document model, without any special machinery to support annotations per se. Annotations with these capabilities are simply a straightforward, if important, application of this document model.

In this paper, we first briefly describe the Multivalent Document model. We then describe a range of annotation types we have implemented within this framework. We classify these different annotations into those involving (1) span of elements, (2) geometric regions, and (3) tree structure.

All these annotations compose together; indeed, sometimes the emergent properties of these combinations are novel and useful, and correspond to intuitive wholes, as we illustrate with an example. We then contrast Multivalent annotations with related work, and discuss future directions we plan to explore.

2. The Foundation: Multivalent Documents

The Multivalent Document (MVD) model [PW96a, PW96b] regards documents as compositions of intimately related but distinct *layers* of content (text, images, video, hyperlinks) and dynamically loaded program objects called *behaviors*. Layers and especially behaviors are expected to be assembled by an MVD-compliant browser from multiple distributed sources over the network. Potentially any media type, including existing document formats from scanned page images to PostScript to SGML, can be bridged into the Multivalent model. Behaviors can be added dynamically, thereby extending the capabilities of document in the browsers. Previously, we have applied MVD to scanned document images; recently we have begun to support HTML as well.

Our application of MVD to scanned document images provides a useful illustration of the concept. In this application, shown in Figure 1, the layers include the document images, the result of submitting these images to a document recognition process (OCR), and possibly other information. Behaviors were implemented to support many of the functions familiar in browsers and word processors, plus some that perform rather novel functions. For example, a "search" behavior implements a conventional text string search. However, because another behavior composed the information available in the image and OCR layers, one can use the search behavior to search for words in the scanned image, where matching terms are highlighted. Similarly, it is possible to select from the scanned image, and paste the corresponding text into another application. These and other behaviors together contribute to the feeling of "enlivening legacy documents".

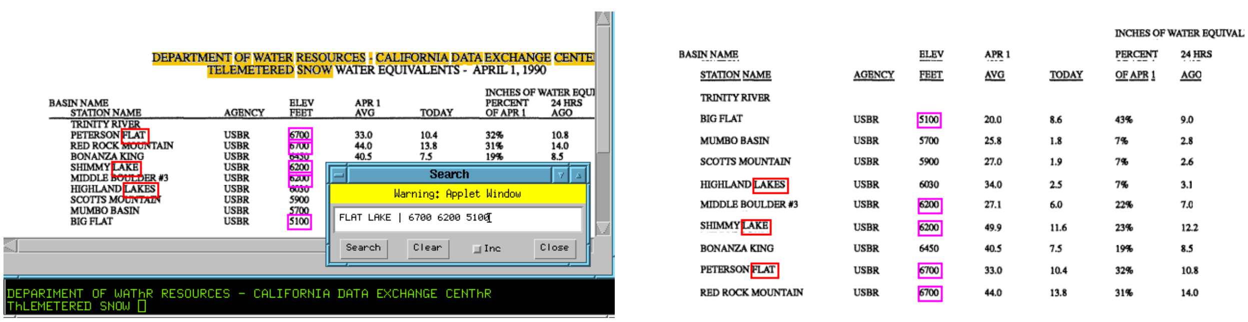

Fig. 1. "Enlivened" scanned page images. Viewing an image of a table on the left, "DEPARTMENT" through "SNOW" has been selected and the corresponding OCR pasted into another application window below; various search term matches have been outlined in the image. On the right, the table in the image has been double-spaced and sorted by the third column.

The MVD architecture has been implemented in Java, and has been applied to the 200,000 scanned page images of the Berkeley Digital Library collection. The reader may examine the functionality described herein by pointing a Java-compliant Web browser at http://elib.cs.berkeley.edu.

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 by several features: (1) The Multivalent Document model defines a 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) there is a separation of structural document content from media-dependent elements, and (3) there is a single coherent abstract tree representation of the document, into which all content is combined and upon which all behaviors operate.

To avoid arbitrary restriction on the extensibility of the system, each of the fundamental runtime operations on digital documents has been opened to a protocol that can be extended. The fundamental document lifecycle—which can be found in some (perhaps abbreviated) form in almost all digital document systems—is conceptualized into protocols beginning with document instantiation (**restore**) into a graph data structure (**build**) which is **format**ted and then rendered on the screen (**paint**), at which point the event loop waits for **user events** from the keyboard, mouse or other input device. Events can trigger a **save**, cause the document to **print**, **select** a portion, or some other action that may require looping back to an earlier phase.

Behaviors modify or extend the system by contributing methods to each protocol. 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 giving higher priority behaviors an opportunity to "shortcircuit" ones lower from the chain, as for instance when a collapsed outline node bypasses formatting its hidden contents for the sake of efficiency. In the up stage, control flows low priority to high, giving higher priority behaviors the opportunity to "massage" the results from lower priority behaviors.

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 contribute to the construction of a document structure tree. The encapsulated media type communicates information such as the bounding boxes of its internal content (at some granularity, e.g., a word of a textual document) and renders that content upon request from the framework. This creates a multimedia document system. Because behaviors operate on the medium-independent structural document tree and communicate to encapsulated media types through the protocols, behaviors are written once without special accommodation for any particular medium and, as much as it applies to a given medium, operate on all media.

Behaviors can contribute items to menus in a conventional menu bar (not shown here, but visible in Figure 3 below), allowing one form of user interaction. Individual behaviors can also intercept user events, as some examples below illustrate.

The architecture, presented here in highly condensed form, is elaborated in [PW97].

In the Multivalent model, "document" refers to the "hub" document, essentially a list (written in a language defined in XML [BS97]) of the layers and behaviors that comprise a single conceptual document. To restore a document, the framework fetches these pieces over the network and composes them together. For example, the hub document of the enlivened scanned image documents alluded to above comprises URLs for the image layer and OCR-produced text layer derived from the images, plus possibly supplementary layers, such as that containing a description of a table, along with supporting behaviors, e.g., one specific to this type which constructs a tree from these layers, plus general behaviors that support searching, etc. Another example of a hub document is a "base" document along with layers and behaviors that impose annotations on this document. We now examine a variety of such combinations.

3. Multivalent Annotations

A decade ago, Halasz's classic "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems" [Hal88] identifies "Extensibility and Tailorability" and "Support for Collaborative Work" with annotations as key challenges. The Multivalent Document model addresses the extensibility concern and provides a base for work on collaboration through annotations. As experimental evidence indicates that users want "to regard annotations as a separate layer of the document ... perceptually distinct from the underlying text" [OS97], the Multivalent Document model is naturally suited to annotations where groups of annotations, as by a single author, are collected together as layers that can, as a central aspect of the system, compose with a "base" document and other annotations.

Multivalent annotations meet many of the requirements we stipulated for digital annotations simply by virtue of operating within the Multivalent Document model. Multivalent annotations are implemented as or by specialized behaviors. As such, they are composable (the Multivalent framework manages them through the protocols), source format independent (they manipulate the abstract document tree and communicate to encapsulated media types through media adapters), server independent, extensible, seamlessly integrable, immediately portable over the network, and powerful (they have access to every state of the fundamental document life cycle).

Another desiderata of digital annotations is that they appear *in situ*. We accomplish this requirement by having the individual annotation behaviors rely upon the geometric placement information of document components 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. All this is managed by the Multivalent framework calling the annotation at the right time.

Finally, we require that annotations be robustly positioned. This requirement is addressed via a general feature available in MVD, albeit one implemented with this purpose in mind (though it could in fact have been implemented as a behavior). Specifically, a standard system class is provided that takes a document structure tree position and creates a redundant description of that place, including its position in the structural tree and offset into the leaf node, an excerpt of the underlying text, a unique identifier of any anchor points, and other information as available. 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 annotation to the new appropriate location. For instance, if a block of text were deleted before the annotation, 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 annotation 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 being preferred to those farther away when several matches produce a choice. If every attempt at reattachment fails, as when the corresponding area of the document is deleted entirely, this fact is reported to the user, who can reattach the annotation manually or discard it. Preliminary results indicate that in practice the annotation repositioning strategy works well; of 754 annotations that needed repositioning to layers that underwent varying degrees of mutation, 742 annotations 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.

We have developed three broad types of Multivalent annotations, i.e., annotative behaviors with associated data conforming to the MVD model, As categorized by their technologies, they are annotations that make use of (1) point-to-point spans of media elements, often characters, (2) geometric regions, and (3) structures within the document tree. We reiterate that these three types are just examples which we have implemented to demonstration a range of possibilities, not built-in characteristics of the MVD architecture.

3.1 Span Annotations

Span annotations are annotations that make use of spans, objects that extend from one point in the document continuously to another point. Spans are implemented with an object attached to its start and end points in the document tree. Span objects can 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. Behaviors can be defined that manage span objects, creating and destroying them at user request through the user interface, and saving and restoring them from persistent storage.

Figure 2 shows a document containing several types of span annotations. This document is a scanned page image, enlivened by an optical character recognition layer and a set of supporting behaviors, as described above. However, unlike the pages in our own repository, this document exists on a server for which we have no write permission; yet we have annotated the document in a number of ways.

Perhaps the simplest annotation example is a digital version of yellow highlight marking. Readers often use a yellow marker to make passages of a paper document visually prominent and hence easy to find again later. As a span annotation, such a highlight would comprise a span of the appropriate text, whose background color had been set to yellow. The text in the middle of the page from “Real disk I/O” to “with a simulation” is just such a highlight span. In this case, a behavior is used to author the annotation. Specifically, the user created this annotation by first selecting a region of text from the image using the mouse, and then choosing a “highlight” behavior from a menu (not shown); this behavior creates a new span corresponding to the selected text, changing the span’s background property to be yellow. (It would have been just as easy to write a behavior that changed the foreground color, XOR mode, underline/overstrike, font, x and y displacement, scaling factor or visibility.)

Note that while spans decorate the structural tree of the base document, they are separate, robustly anchored components that may be stored independently of the base. Thus, restoring an MVD hub document corresponding to the annotated document would result in assembling the annotated document from the unmodified base layer at one source, and the highlight annotation from another, and building the tree whose rendering results in the appearance of an *in situ* annotation.

Another example of a span annotation is a hyperlink. While hyperlinks may of course be supported internally by a document format, as they are in HTML, one might want to annotate someone else’s document with an additional hyperlink, regardless of whether the document format supports this structure. In MVD, a hyperlink can be implemented as a span annotation that utilizes mouse events. Figure 2 exhibits such a hyperlink, authored on a scanned page image. Specifically, the span across the terms “fault tolerance” is a hyperlink. The behavior managing this span will cause the user’s subsequent interactions with the span to exhibit the expected results, i.e., dragging the cursor over the span will show the destination on the bottom of the screen; clicking on the span will follow the hyperlink. The same behavior was also used to create the link, analogously to how the yellow marker highlight was created, except that here the user was asked to supply a URL, and the span's underlining feature has been turned on. The selected span "IBM Risc System 6000 Model 350" is shown in the process of being made into a hyperlink, a process which would be completed with a click on OK. Because hyperlinks are just another extensible behavior in the system and not a core fixed feature, hypertext researchers are free extend the system and experiment with different types of hyperlinks, such as typed links [Trig83], with a very small amount implementation effort.

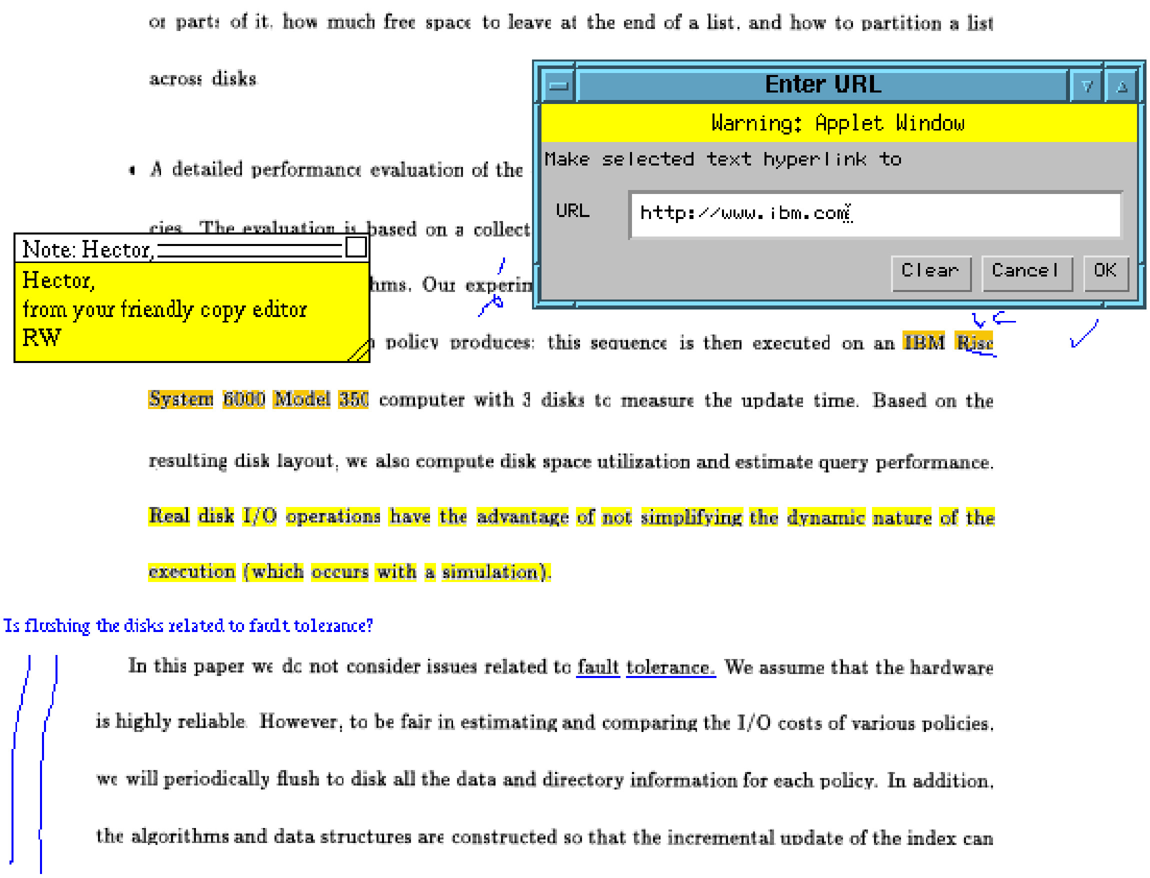

Fig. 2. Span annotations: highlight, hyperlink, copy editor markup. The background of highlighted text (from "Real disk" to "a simulation" is yellow; the hyperlink (spanning "fault tolerant") is a colored underscore; the background of selected text (spanning "IBM Risc System 6000 Model 350"), which is about to be turned into a second hyperlink, is in another contrasting color; the other marks on the page are blue copy-editor comments. (The Post-It-style note is an example of a geometric region annotation, described below.)

Another, more complicated example of span annotations is copy editor markup. Blue copy edit marks are seen scattered about the page. Some are freehand sketches; others contain typed text; still others arise from the application of a palette of common markup symbols. In the latter cases, readers can automatically "execute" the markup (an option that is likely to be more useful for formats other than scanned images).

Another example of an implemented span annotation, but one not illustrated here, is a behavior that visualizes version differences, with deleted text overstruck, inserted text in italics, and changed text in bold italics. While not new in concept, this function is implemented not as a special feature of the model but as yet another behavior, one that modifies the content of the runtime document.

All span annotations, even more geometrically oriented ones like sidebars, are attached to words or other media elements rather than to fixed x,y coordinates on the page. In this way, when the document is reformatted, the annotations can be repositioned correctly. Indeed, Figure 2 illustrates a scanned page image after the application of a behavior that implements double-spacing. While, double-spacing is an independently motivated behavior, it is especially useful in conjunction with annotations, as it opens up real estate for their use. Note that the highlights, hyperlinks, et cetera, follow along with the referenced portions of the text as the document is double-spaced. Moreover, they do so without any special effort by the programmer of those annotation types (or of the double-space behavior).

As promised, span annotations are robust against changes to document content as well as document manipulation. During editing, layers are mutually tied together with “sticky pointers” [Lad], which maintain perfect alignment. Span annotations are deleted when the entire span being annotated is deleted, and expand to cover any text inserted into the span. When saved to persistent storage, spans use a system class to achieve gracefully degrading repositioning, as described above.

3.2 Geometric Region Annotations: Lenses

Spans allow for the association of annotations with the fine-grain structure of a document. Another type of Multivalent annotation, *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 3 illustrates two kinds of lenses, operating together with others behaviors. To the left, in the “Show OCR” window, the image text is replaced by the results of an OCR process, rendered in the font the OCR software estimates for the original text.

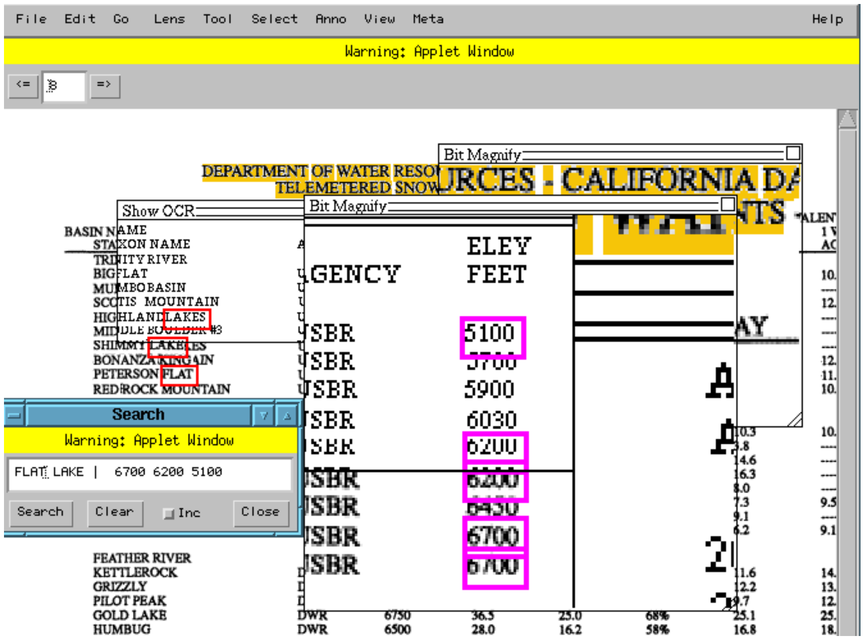

Fig 3. Geometric or lens annotations: One “Show OCR” and two “Bit Magnify” lenses are shown, in composition in overlapped regions, and with the results of a search behavior and a behavior that sorted the image by the “ELEV FEET” column.

The upper right “Bit Magnify” lens enlarges the image underneath. A second “Bit Magnify” lens in the middle, on top of the other two, magnifies as well. Where the lenses overlap, the effects compose. The upper left of the second “Bit Magnify” lens overlaps the “Show OCR” lens, and so enlarges the OCR translation; in its lower right, it overlaps and enlarges the base scanned image. In its upper right, it overlaps the other magnification lens, resulting in double magnification, and when the other magnification lens in turn overlaps the “Show OCR” lens, the result is double magnification of OCR text.

This somewhat baroque (but realistic) example illustrates that lenses compose with each other. Of course lenses compose with other types of behaviors as well. For example, the terms in the search window are highlighted in the document proper, as well as in the lenses. That is, the search behavior composes with the various lenses. A more dramatic example is the “table sorting” behavior. This behavior sorts tables by manipulating the scanned image. In this example, a click on the table heading of “ELEV FEET” has sorted the table in ascending order of the contents of this column. Both the user interaction and the result compose through the lenses.

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 can arbitrarily transform the appearance of their contents, with full access to the semantic representation of that content. 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, even inside stacked magnify lenses.

Another type of lens, an opaque one, was used to implement the movable note shown along with span annotations in the previous figure, Figure 2. The window apparatus is shared with other types of lenses. The only difference is that during the down or high to low priority phase of painting, the lens-note paints its background and then short-circuits any lenses below.

We have experimented with a “language translation lens”. It takes as additional layers sentence-aligned translations (constructed manually). When positioned over a line of a sentence, it displays underneath that line the corresponding translations, in our case French or German.

Most lenses transform the appearance of content they contain. In this they share the same graphics context as span annotations. In fact, it is easy to construct behaviors that permit the user to dynamically define, via a lens or span editor, new appearance-transforming lenses and appearance-transforming span annotations. The user would set the desired appearance transformations from a panel of these attributes. All media elements would then draw themselves according to these prevailing settings. (Reacting to events presently requires coding in Java; on-the-fly programming would require a yet-to-be designed scripting language.)

The list of standard display parameters—foreground color, XOR mode, underline/overstrike, font, x and y displacement, scaling factor and visibility—does not cover the possible desirable data type-specific transformations. (For example, it does not permit the “show OCR” transformation.) For such cases, lenses also carry 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 lens on top.

3.3 Structural Annotations

Structural annotations hook into the basic document tree, which is hierarchically structured (e.g., a book contains chapters that contain sections that have subsections). Whenever action is happening in that area of the document, structural annotations are given an opportunity to modify the results. They can invest incremental effort into a document or leverage existing structure.

As an example, 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 4, a bibliographic entry has been selected. To incorporate this entry into a database, one could start by pasting the OCR text and editing it as necessary. Instead, we have created a “Biblio” behavior that automatically performs the desired transformation. 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 refer-formatted text, as the user chooses. Once in the selection buffer, of course, this formatted text could be pasted into any application, as evidence in the text windows on the right of Figure 4. These formatted text is computed on the fly, so that adding an additional output format merely requires coding the appropriate formatting statements.

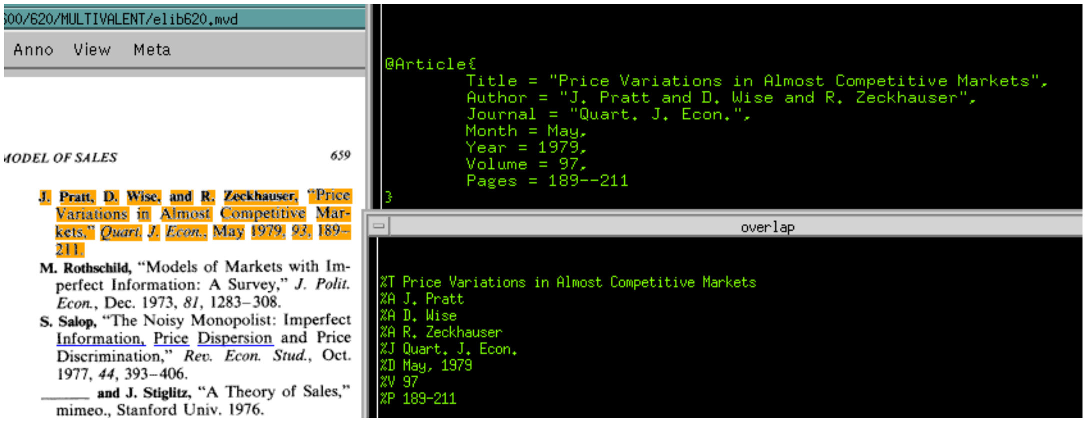

Fig. 4. Select bibliographic region from scanned page image, paste corresponding BibTeX and refer, according to menu setting.

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

3.4 Combining Annotations: NoteMarks

Paper documents can be laid out in space to gain a sense of overall structure and to cross-reference specific information across pages [OS97], distinct advantages over the digital medium. While computer hardware may not approach paper's flexibility in the near future, an orchestrated combination of Multivalent annotations can exploit computer processing of the document to recover some ground.

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]. Here the same functionality has been duplicated in MVD, as illustrated in Figure 5. First, structural annotations are used to allow a user to collapse or expanded a section by clicking the section header. However, within otherwise collapsed outline sections, single lines can be visible. These lines are controlled by span annotations of higher priority than the structural annotations, and thus can override the visibility state. Some such spans may be 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. In Figure 5, note that, in the case of subcommands, command line options and excerpts, the system automatically located the text and applied the span annotation. The highlights, applied by the user, and the search hits, found by the system at user's request, also override the collapsed view of the section.

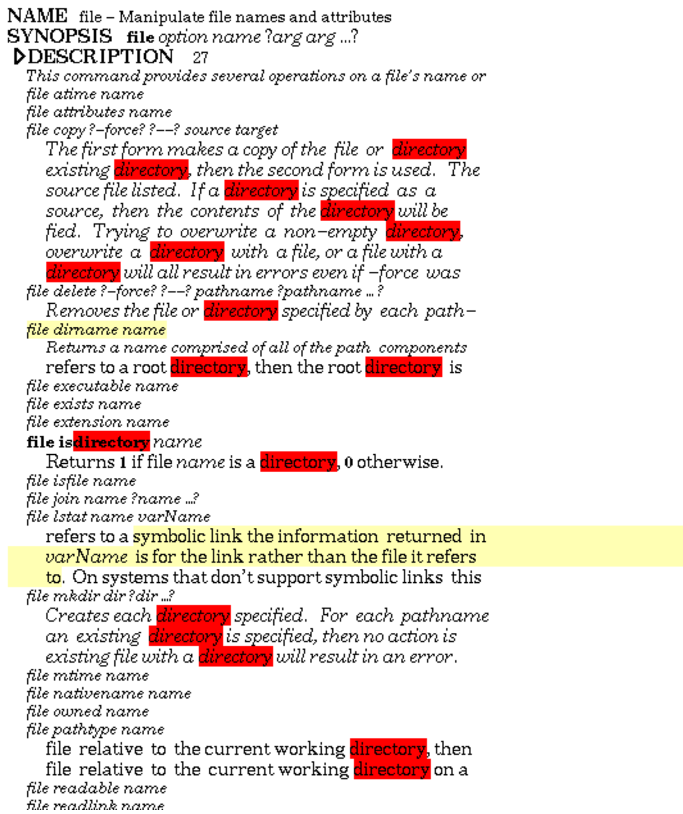

Fig. 5. NoteMarks: Within otherwise collapsed Description section, subcommands, highlights and search hits show through. An implicit hyperlink covers the text so that clicking on the desired area opens up Description and scrolls to that line.

A NoteMark is note-like, in that the collapsed-but-overridden format may comprise a useful summary; it is like a bookmark, in that clicking on a NoteMark causes it to act as an implicit hyperlink, automatically expanding and scrolling to the corresponding section. In short, NoteMarks provide a customized display of the overall structure of the document, as if one folded a paper document to display desired portions. Note that NoteMarks is not a specialized capability of the Multivalent framework; it is simply a combination of several different kinds of annotation.

4. Related Work

One feature of the Multivalent model that supports annotations is extensibility. Many other forms of document extensibility have been devised previously. For example, HTML with Java applets, OpenDoc [Appl95], and OLE [Broc95] are extensible but only at a relatively coarse-grained level. Each new bit of functionality receives its own plot of screen real estate, and greater complexity is built up by juxtaposing plots one next to another. While there is some mechanism for interapplication communication, it not generally possible to augment another part, adding a hyperlink or a lens say. That is, parts cannot compose; each applet/part/object has a circumscribed region of activity. That makes these systems unsuitable as the basis for an annotations system.

Adobe's Acrobat [Adobe] 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, unlike early HTML viewers, 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. If one hopes to accommodate innovative new technology, it is unlikely that these hooks will indefinitely prove sufficiently versatile.

ComMentor [CMW95] 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 could only provide minimal functionality at the client as it relied on taking the source code for Mosaic and enhancing it; Mosaic was a moving target and has now been entirely superseded by commercial browsers—whose source code is not available. ComMentor 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 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 exist in recognition 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, annotation requirements push against the limits of such systems. Instead, we pursued a strategy makes it more difficult to take advantage of existing applications, but, we hope, will prove sufficiently general to enable the incorporation continuing innovations in digital document technology.

5. Future Work

Here we have focussed on scanned page images, showing how MVD can be used to enliven legacy documents. Other document formats are far more widespread. In particular, we believe that distributed annotations operating on top of HTML would be widely useful, and are applying the model to HTML.

An important missing component of the model with respect to annotation is good authoring tools. We have shown some examples above of authoring behaviors (e.g., tools that create a highlight or a hyperlink), but widespread use of more complex annotations types will require better tools with good user interfaces.

Although the current system implementation demonstrates the research claims, we plan to invest much work in system robustness, usability testing and performance tuning.

With layers and behaviors generally but especially with annotations, one would like to take a document and globally collect commentaries on it (i.e., retrieve all annotations by various authors).

We are defining an annotation identification system so that a user can employ a generic Web search engine to identify applicable materials.

6. Conclusions

Multivalent annotations is a format independent, extensible, networked form of annotations based on the Multivalent Documents model. They provide an open-ended set of ways to mark up a digital document, even one kept on a foreign server, and to transform the document, some of which we have illustrated; no doubt there are numerous others that we have not even anticipated. Perhaps Multivalent annotations do not make digital documents as inviting as paper, but they move the digital medium one step closer.

Acknowledgments

Loretta Willis implemented the table sorting behavior, patterned after an earlier prototype. Pascal Mueller-Gugenberger implemented the French-German language translation lens, patterned after two existing behaviors.

7. References

[Adobe] Adobe Systems. Acrobat. Commercial software.

[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.

[CMW95] 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: Technology, Tools and Applications*, April 10-14,1995.

[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.

[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.

[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.

[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.