Designing BiblioText: An Experiment in User Interface Design

Michael L. Van De Vanter[1]

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

October 24, 1988

Report No. 88/454

PIPER Working Paper 88-1

ABSTRACT

Principled approaches to the design of user interfaces are typically unsubstantiated by practical experience, or are validated only through laboratory experiments with toy programs. This experiment applies several theoretically motivated approaches to the user interface of an existing, practical program of moderate complexity; these approaches include conceptual models, task based design, the use of metaphor, direct manipulation, and empirical evaluation. Although necessarily informal, the experiment yields insight into the utility of these approaches, as well as observations on more general themes such as specialization versus generality, multiple models and their limits, the problem of context, and the elusiveness of consistency as a design goal.

---

[1] Sponsored by the Defense Advanced Research Projects Agency (DoD), monitored by Space and Naval Warfare Systems Command under Contract N00039-88-C-0292, by IBM under IBM Research Contract No. 564516, and by the State of California MICRO program. Michael L. Van De Vanter was supported in part by a MICRO fellowship.

Table of Contents

1. Introduction

The research literature on human-computer interaction is filled with theoretically motivated approaches to the design of user interfaces. Unfortunately, these fall too often into one of two categories. Contributions in the first group present principles, supported by psychological theories and persuasive arguments; these are never based on experimental evidence, and are seldom supported by experience with working, practical systems. Fred Brooks calls contributions in this category *rules-of-thumb*[2] [Bro88].

Contributions in the second category support proposed general principles with controlled experiments. Fred Brooks would call these contributions *findings*[3]. Unfortunately the supporting experiments take place typically in laboratory settings, with naive users and with artificially small (toy) programs. This necessarily bounds rather narrowly the relevance of the results to the problems of larger systems, despite occasional claims to the contrary.

This paper reports an experiment that takes the middle ground, what Fred Brooks would call *observations*[4]. Although less formal than laboratory results, the results of this experiment are considerably more relevant to the problems faced by those who design user interfaces for real systems.

The experiment applies several theoretically motivated approaches to the user interface of an existing, practical program of moderate complexity. These approaches include conceptual models, task based design, the use of metaphor, direct manipulation, and empirical evaluation. One result of the experiment is an evaluation of the benefit of the different approaches.

Other results of the experiment are more general observations, resulting from the meeting of theory and practice. General themes that emerge from the experiment include tension between specialization and generality, multiple models and their limits, the problem of context, and the elusiveness of consistency as a design goal.

The experiment took place in three phases:

- *Phase I.* Evaluate the current user interface of an existing program, using a variety of theoretically motivated approaches (reported in sections 3 and 4).
- *Phase II.* Redesign the user interface, based on suggestions arising from Phase I (reported in section 5).
- *Phase III.* Evaluate empirically the resulting interface, and judge the benefit of the suggested improvements (reported in sections 6 and 7).

The report begins with an introduction to *BiblioText* [Van88], the experimental subject drawn from the universe of programs (section 2 "Some Background"). BiblioText is an interactive, window- and mouse-based program for browsing collections of notes, papers, and bibliographic reference data. It is a byproduct of ongoing research on the design of personal information management tools and on issues in user interface design.

---

[2] "... generalizations, even those unsupported by testing over the whole domain of generalization, believed by the investigators willing to attach their names to them."

[3] "... those results properly established by soundly-designed experiments, and stated in terms of the domain for which generalization is valid."

[4] "... reports of facts of real user-behavior, even those observed in under-controlled, limited sample experiences."

This research is part of the PIPER projects at UC Berkeley. Section 2 describes BiblioText's evolution from prototype through version 3.3, its form at the beginning of this project. The report refers to this version as *the old interface*.

The first phase of the experiment begins with an evaluation of the old interface (section 3 "Descriptive Models"). This section presents, among other methodical descriptions, a newly developed conceptual model for its operation. The results of section 3 provide the basis for the evaluation of the old interface at higher levels of abstraction (section 4 "Theoretical Evaluations"). These evaluations draw on approaches to interface suggested by the research literature on user interface design, including task-based design, conceptual models, metaphor, and direct manipulation. The analysis based on each approach suggests potential improvements to *BiblioText*'s design.

The second phase of the experiment redesigns the user interface for BiblioText (section 5 "Redesign"), based on suggestions for improvement originating in the first phase. Section 5 discusses some of the conflicts and tradeoffs involved in resolving the various suggestions. Many of the suggested changes were made, producing BiblioText version 3.4, referred to as *the new interface*.

The third phase of the experiment begins with an evaluation of the new interface, based on two sessions during which new users were introduced to the program (section 6 "An Empirical Evaluation"). This section summarizes those results and discusses apparent trouble spots in the interface, some that had been predicted and some that had not. The third phase continues by using these empirical results to judge the changes made in the new interface (section 7 "The New Interface Reconsidered"). The discussion in section 7 leads to further suggestions for improvements to the *BiblioText* interface.

Most sections conclude with a few thoughts and insights that resulted from that step of the experiment. The final section of the report (section 8 "Conclusions") summarizes these specific results and considers some general themes that arose throughout the experiment.

2. Some Background

BiblioText is window- and mouse-based program for browsing bibliographic data and related notes [Van88]. It is intended for a working environment characterized by indexed bibliographic data in *bib*, *refer*, or *tib* format, along with online documents containing imprecise citations that point into the database.

At the time this experiment began, BiblioText had evolved from an early prototype into a working, practical program. It had been in daily use for over a year. This section reviews a bit of BiblioText's history, establishing some of the context for the experiment.

2.1. The First Prototype

The BiblioText prototype emerged from a small project in user interface design. The goal of that project was to explore the window/mouse environment of the workstations that were beginning to appear. The general approach was to port into the window/mouse paradigm the functionality of some existing batch programs, and to examine the success and implications of the shift.

The Task Domain

The domain of the project derived from my interest in tools for professionals, working alone or in small groups, who manage and selectively share personal information. This information includes bibliographic references, on-line documents, personal notes, and their mutual interconnections.

I began with a look at the task domain, describing a prototypical working environment and listing typical tasks in that environment. The analysis (summarized in appendix A, "Original Task Analysis") discusses how a person might accomplish each task in the absence of any automated tools. This analysis was the basis for the prototype design (see below) and appears again during part of the experiment described in this report (section 4.2 "Task Analysis").

The Functional Design

In a second look at the task analysis, I examined how each task might be partially automated using existing (batch-oriented) software for document preparation in the UNIX[5] environment. That environment includes text editors (*vi* [Joy79] and *emacs* [Sta81]), a typesetting program (*troff* [Oss76]), and a suite of tools (*bib* [BuL82]) that specifically addresses both the maintenance of bibliographic data as well as the production of citations and references for inclusion in *troff* documents.

This second analysis revealed a niche in which an interactive program could be of assistance, and suggested the functional design of an interactive browser for bibliographic references and documents. The functional design had five major components: collect references (by keyword lookup or by document preview), display the current collection, store the collection, manipulate the collection, and display useful messages. That design, supplemented by a few pages of detail, was the only formal account of BiblioText's behavior until the first phase of the experiment described in this report (section 3.2 "A Conceptual Model").

Design Constraints

The functional design left considerable latitude for the detailed design and construction of the prototype. The following constraints affected the development of the prototype at least much as the functional design.

- *Delivery Deadline.* Little time was available to produce a working prototype.
- *Existing Tools and Data.* The browser had to share data with existing tools, notably *bib*.
- *Programming Support.* The prototype was implemented on a Sun Workstation[6]. This choice maintained compatibility with the existing UNIX environment, but demanded reliance on release 1.1 of SunView software libraries [SSS84]. The quality of the early SunView release[7] imposed serious constraints on the quality of the prototype.

---

[5] UNIX is a registered trademark of AT&T Bell Laboratories in the USA and other countries.

[6] Sun Workstation and SunView are Trademarks of Sun Microsystems, Inc.

[7] Early releases of SunView were called SunWindows. Many of SunView's limitations disappeared with the later releases, but some are inherent.

Major Design Decisions

Given the constraints of the project and the available tools, several major design decisions emerged.

- *Represent bibliographic data in* bib *raw format.* This made available a large pool of data and potential users. It had the further advantage that existing software could be adapted for use in the prototype. It had the disadvantage that constructing *bib* indexes is computationally expensive; hence the following decision.
- *Only allow the browser to* read *bibliographic data.* The incremental maintenance of index structures is a formidable database problem. This did not prevent, however, changes to documents from within the browser (to the extent that indexes remained valid).
- *Undertake no other major software modifications.* Time did not permit integration with text editors, any elaborate low-level programming in SunView, or modifications to the basic algorithms in *bib* programs for indexing, retrieval, and the like.
- *Add features as quickly and cheaply as possible; rework the interface later for usability.* Time was short, and the project was an exploration, both of *bib* functionality and of the interactive programming paradigm. Experience suggests that commercial workstation software is often (if implicitly) developed this way too[8].

Problems with the Prototype

Like most prototypes, the first BiblioText was unsuited to real use. Many items in the functional design, including some of the most interesting, were left for "further work." Implementation shortcuts caused the prototype to run unacceptably slowly. Finally, the user interface was hopelessly confusing. The control panel contained a bewildering array of objects for poking and typing into; they amounted to little more than "forms" for UNIX command line options inherited from batch programs. Many were arcane, even beyond the expertise of most *bib* users. The project report included a sketch of a better user interface, but it relied on software support not yet available in SunView.

2.2. Evolution

In spite of its limitations, the prototype demonstrated the potential utility of the browser. During the year and a half between construction of the prototype and the experiment described in this report, I developed and used BiblioText, both as a personal tool and as a testbed for minor experiments with user interface design.

Efforts to improve the browser were sporadic, responding to a variety of motivations. First, as I used the program, I began to understand which functions and which parts of the user interface (assembled originally by the "kitchen sink" method) were important. Second, as the browser matured, I gradually changed my personal data and work habits to take advantage of it. Finally, several successive releases of the SunView window system changed the implementation environment considerably, in many cases

[8] Like many other projects, the anticipated "rework" did not happen, other than evolutionary change (until the experiment described in this report).

presenting further opportunities for improvement.

The rest of this section describes these changes under the three headings: internals, functionality, and user interface. Unfortunately, experience showed that this separation is artificial. More often than not, a change of one kind precipitated changes of other kinds.

Internal Changes

The most urgent internal changes produced better internal data structures and incremental algorithms; these gradually replaced the background batch jobs used by the prototype. Internal data structures and algorithms continued to evolve in response to later functional changes. In several cases, new SunView releases demanded restructuring just to keep the program working.

Functional Changes

I gradually completed the functionality sketched in the original design, but only as I needed it. Some major functions changed gradually, as their roles in the new incremental setting became clear. For example, it turned out that bibliographic references in BiblioText's viewer need not be formatted subject to numerous options (in the manner of the batch programs *listrefs* and *troff*). Reference formatting eventually contracted into a single method with two binary switches; these seemed to give the user just the right amount of control.

Along the way, several minor functions design became obsolete. Some were simply subsumed by other, evolving functions. In other cases, they became irrelevant. For example, some of the original minor functions turned out to be artifacts of a particular style of user interaction; when that style changed, the functions disappeared.

Some of the originally designed functions never did seem important enough to implement, although one resurfaced later, during the experiment described in this report.

New functionality emerged as I integrated BiblioText with the evolving SunView environment. For example, the SunView global selection service made it possible to cut and paste between windows, to the advantage of some BiblioText functions. The SunView defaults database arrived to provide a more powerful and simpler method for allowing user configuration; this replaced the original ''.bbrc'' file that contained startup information.

User Interface Changes

I gradually simplified the ''kitchen-sink'' control panel; this nearly always required other changes, both to internal structures and to functionality. For example, the sorting function originally required that the user type a sort ''template'' according to an obscure convention. When SunView menus arrived, I replaced the template field with a button and popup menu; the menu displayed a predefined set of sorting options, labeled in plain English.

The arrival of secondary frames made it possible to implement popup prompters for short interactions, eliminating more type in fields. They also made it possible to display auxiliary information in auxiliary windows, eliminating the need to multiplex the main

window (and thereby eliminating a mode).

Finally, BiblioText's appearance improved gradually, in no small part because the SunView library of objects did likewise.

2.3. Thoughts on Program Design and Evolution

Experience with the original project at subsequent evolution anticipated some of the issues that appeared during the experiment described in this report. Some of these were:

- *The Paradigm Shift.* The prototype imported both functionality and software components directly from existing batch programs. Many proved to be generally incompatible with the needs of the browser, which follows the *editor paradigm*; most of them eventually changed.
- *The Representation Problem.* Weaknesses in the *bib* data model become gradually more troublesome in this new environment.
- *Implementation Support.* Inflexibility in early SunView software made it impossible to exploit the full potential of the workstation.
- *Information Structures & Hypertext.* In the original report I began to think about BiblioText as "a specific instance of a hypertext browser." This line of thinking guided many subsequent changes, before and after the experiment described in this report, and emerges as a main theme in the current version of BiblioText [Van88].
- *Coevolution.* It would be more accurate to describe changes to BiblioText as coevolution together with my personal working environment; both changed gradually with a great deal of influence on one another.

3. Descriptive Models

This section begins the first phase of the experiment, an evaluation of BiblioText as it existed at the beginning of the project. This evaluation is based on version 3.3, referred to in this report as *the old interface*; figures 3.1 and 3.2 below show the old interface in operation.

The evaluation of the old interface begins with the construction of a set of descriptive models that explain various aspects of the program. The models reported in this section are the starting point for the higher level evaluations that follow (section 4 "Theoretical Evaluations").

During the construction of these models, the process of putting aspects of BiblioText into various coherent frameworks turned out to be a productive form of analysis itself. Several issues arose that foreshadowed later problems; the concluding part of this section reports some of these, along with reflections on the utility of building these models in the first place.

The following parts of this section discuss four different descriptions of the old interface. The first was to be a complete specification, to serve as the basis for later models and later phases of the experiment. This attempt ended in failure, swamped by details. The second description is a predictive model; it presents a simplified view in terms of conceptual objects and operations, with as much interface detail as possible excluded. The third and fourth describe aspects of the user interface: state information

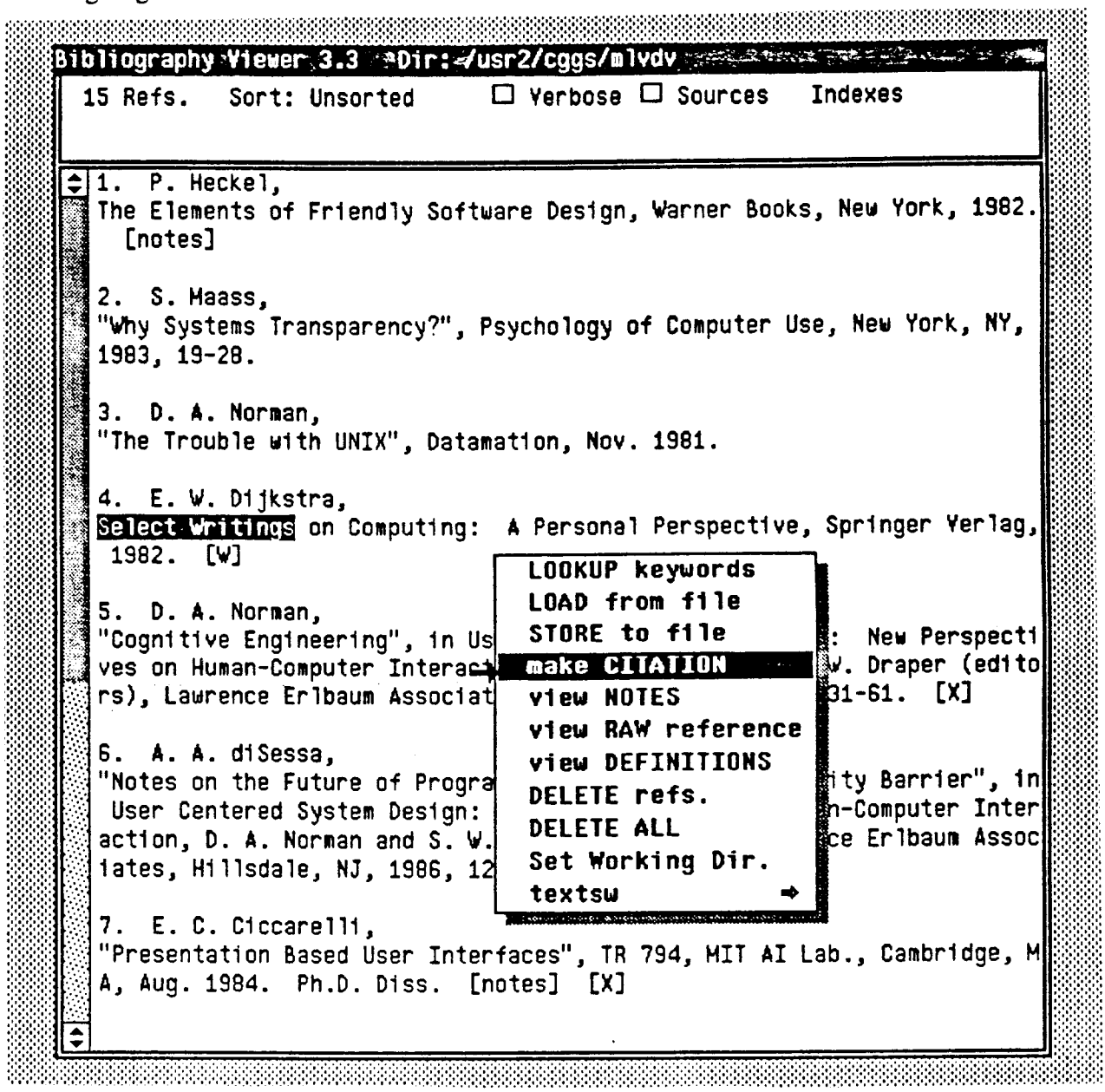

Figure 3.1
The old interface: main menu visible

present inside BiblioText (and the visibility of that state), and an inventory of user commands and their syntactic categories.

3.1. A Full Specification

The first description of the old interface was to be a complete functional specification of BiblioText's old interface. The specification would describe every aspect of the program's operation, as seen by a user. It was intended to serve as the baseline for anticipated design changes, as well as the foundation for other, more abstract models.

After some effort, this attempt was abandoned. The volume of detail required to give a complete account of the program's behavior grew to the point that the specification became unmanageable and incomprehensible (and therefore useless).

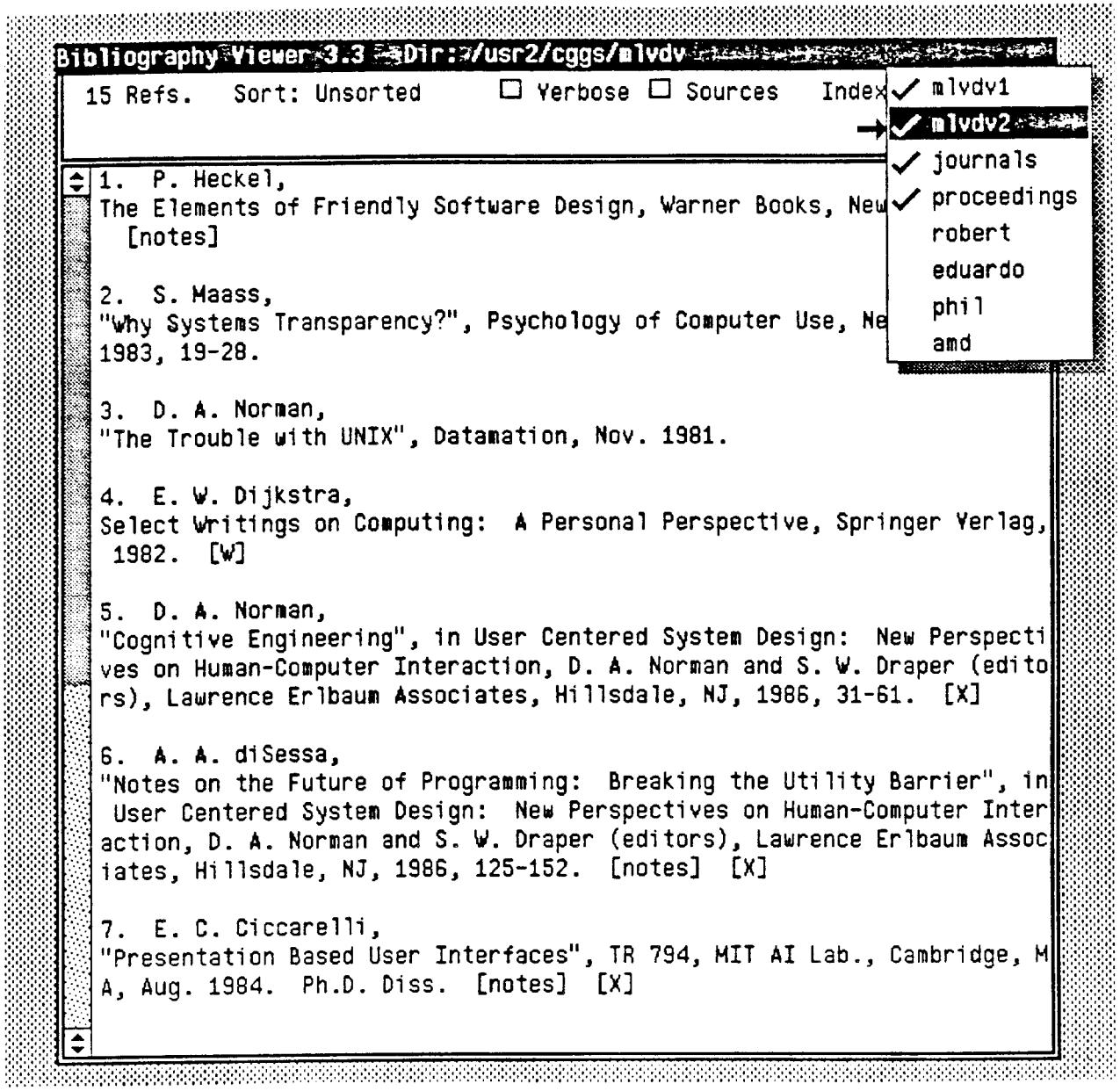

Figure 3.2
The old interface: index menu visible

One problem was the importation into BiblioText of the semantics of various aspects of its operating context: the UNIX file system, the precise semantics of *bib* databases, the physical mouse and keyboard, and interactions with other clients of the window manager. Another source of imported complexity was the SunView toolkit for building interactive programs; to describe precisely the operation of BiblioText would have required the inclusion of better documentation on SunView's operation than was available from its authors.

And yet, none of those imported semantics, with all their complex behavior and rich internal state, could be omitted from a complete functional specification. The final result of this effort was a sketchy inventory, in outline form, that was too long and yet omitted too much detail. The conceptual model described next turned out to be much more useful.

3.2. A Conceptual Model

This model explains what BiblioText does in terms of conceptual entities and operations on those entities, much like the *implied register models* of Young [You81]. Like Young's models, this model enables the user to predict the program's behavior in response to various inputs. Other terms for this kind of model are *surrogate model* and *structural model* [diS86].

This model omits as much mention of BiblioText's user interface as possible. Unfortunately the distinction between conceptual operation and user interface proves to be a blurry one, for many of the same reasons that the attempted full specification failed. The implementation itself offers little help. The boundary between BiblioText's code and that of the imported SunView toolkit library represents more a division of labor and a standardization of parts, not a division between functionality and interface. Both parts of the attend to both areas.

In the end, the choice of how much to include in this model is a matter of judgement. The criterion concerns the utility of the model as an explanation, which is, in turn, dependent on the audience.

I: The Ref.

The atomic concept in this model is the *ref.*, an internally stored copy of a bibliographic reference in *bib* raw form [BuL82]. A typical raw ref. is a group of textual lines (or *fields*), something like the following:

```
%A Douglas C. Engelbart
%A William K. English
%T A Research Center for Augmenting Human Intellect
%J FJCC
%P 395-410
%D 1968
```

Along with each ref. in raw form, BiblioText keeps track internally of the following associated information:

- *Ref. Source:* This is the name of the file from which the raw ref. was read (not the name of the index used to find it).
- *Cite Source:* If a ref. in the collection was located via imprecise citation in a document, this annotation contains (a) the name of the document file containing the citation, and (b) the citation string itself.
- *Separate Notes:* This is the name of a file, appearing in a %Z field.

A ref. may be discarded from the running program, but it does not modify its original source.

II: The Collection

The principal piece of interesting state in BiblioText is the (single) current collection of refs., referred to here simply as *the collection*. It is a list of refs. that contains no duplicates.

III: The Viewer

BiblioText's *viewer* is a scrollable text subwindow that constantly displays a *formatted* version of the current collection. It also displays:

- the sequence number of each ref. in the collection;
- the total count of refs. in the collection; and
- a scrollbar bubble that suggests which part of the formatted collection is currently visible.

The viewer has two binary mode settings:

- Verbose Mode, and
- Sources Mode.

Formatting converts the contents of raw form refs. to a more readable display. Some fields (such as title, %T) are always included in the formatted display. Some fields (such as extra keywords for indexing, %K) are never formatted. Three types of fields are handled specially, using various forms of ellipsis:

- %W (location) always appears as [W].
- %X (additional comments) appears as [X] when Verbose Mode is off and as [X] <some comments> when Verbose Mode is on.
- %Z (name of file containing additional notes) always appears as [notes].

Raw format refs. typically contain abbreviations that are expanded during formatting. For example the string PROC expands to Proceedings when Verbose Mode is on and to Proc. when Verbose Mode is off. Finally, when Sources Mode is on, each formatted ref. is annotated with filenames representing the source of the raw ref. and the source of the citation (if any) which identified the ref.

Operations that affect the viewer are:

- Scroll Viewer relocates the visible part of the formatted display.
- Resize Frame alters the size of the visible part of the formatted display.

IV: The Selection

The user may select a contiguous stream of characters in the formatted display, using standard SunView mouse commands (left button selects a character, middle button extends the selection). The current character selection appears highlighted in the viewer.

The current character selection determines the *current selection* of refs. This is defined to include any refs. whose display contains selected characters.

V: Adding Refs.

The user may acquire additional refs. from outside the collection. In every case, BiblioText appends newly acquired refs. to the collection and eliminates duplicates.

- Lookup from Keywords collects refs. using an index search on the conjunction of specified keywords.
- Read from File collects refs. by reading raw refs. or citations from a specified file.

There are several modes (settings in the browser) that affect these operations:

- *Active Indexes* specify which indexes to use for keyword lookups.
- *Last used keywords* appears as the default keyword argument.
- *Last used input file name* appears as the default file name argument.
- *Current Working Directory* specifies a path prefix for file names.
- The *File Type* of the input file for a Read from File may be raw format refs. or a text file containing *imprecise citations*; the Read function classifies each input file automatically.

VI: Deleting Refs.

The user may selectively removed refs. from the collection.

- Delete Selected Refs. removes the currently selected refs. from the collection.
- Delete All Refs. removes all refs. from the current collection.

VII: Rearranging the Collection

The collection always has the property *Current Sort Order*. One distinguished order is Unsorted, meaning that the order is uncertain.

- Sort Collection arranges the collection into a specified order.

VIII: Saving the Collection

The user may copy the current collection to a file.

- Store to File writes current collection in raw form to a specified file.

There are several modes (settings in the browser) that affect this operation:

- *Last used output file name* appears as the default file name argument.
- *Current Working Directory* specifies a path prefix for file names.

IX: Popups

Popups are auxiliary frames, separately manipulable window entities, that appear transiently for special purposes. They appear in two basic versions:

- *Prompters* appear during command execution to gather additional information from the user. A prompter receives all user input and remains visible until the user presses the left button over either of the panel items OK or Cancel on the prompter.
- *Auxiliary displays* appear in response to commands that request additional information. An auxiliary display persists as a separately manipulable window object until

the user specifically requests that it disappear.

X: Auxiliary Information

The user may request additional information that is not normally visible.

- Make Citation creates a unique imprecise citation for the first ref. in the current selection, using heuristics and inefficient repeated search of the indexes. It makes the result the current primary selection in the SunView selection service.
- View External Notes on Selected Ref. displays for viewing and editing external notes associated with a ref., as identified by a %Z field containing a file name.
- View Definitions displays the files that contain definitions of abbreviations used in raw *bib* data, also known as *bib* macros.

XI: Setting Modes

These commands explicitly set modes that affect the operation of other functions.

- Set Working Directory supplies a path name to be prepended to all file names supplied for reading and writing.
- Set Display Modes sets the two display modes Verbose Mode and Sources Mode.
- Set Active Indexes controls which of the available indexes will be searched during ensuing Lookup from Keywords operations.

XII: Misc.

At startup, the programs reads configuration information from the user's *defaults database*, a standard SunView facility. The user may view and alter this information using the SunView program *DefaultsEditor*. Configuration information includes

- Initial settings of the two display modes;
- Locations of files containing definitions; and
- Descriptions of Indexes to be used for keyword lookups.

3.3. State and Visibility

Unlike the conceptual model presented above, the description here deals more directly with the user interface. This summary lists each aspect of the internal state of BiblioText that is of relevance to the user, and describes how BiblioText makes that state visible. Figures 3.1 and 3.2 show how some of this appears.

Always Visible State

A formatted version of the *current collection* is always visible, at least in part, limited by the size of the viewer and the current scrolling position. The most important parts of each raw ref. in the collection are always visible in the formatted display:

- basic bibliographic information such as author(s), title, journal, report number, volume, date, page numbers; and

- elided versions of some fields, such as [W] for %W (location) fields and [notes] for %Z (location of external notes) fields.

Other aspects of the current collection are also continually visible.

- A *sequence number* (relative to the current collection) appears with the display of each ref.
- The scrollbar bubble displays the current *scroll position*. indicated
- The *number of Refs*. in the collection appears in the upper left corner of the panel.
- The current *text selection*, if any, appears highlighted with reverse video (though it may be invisible because of scrolling).
- The current *Sort Order* is visible in the Sort: item in the panel, top row second from left.

The state of the two display modes is part of the viewer, not of the collection.

- The status of Verbose Mode appears in the top row of the panel as the item Verbose just to the left of a small box; when the mode is on, the box displays a check mark.
- The status of Sources Mode appears in the top row of the panel as the item Sources just to the left of a small box; when the mode is on, the box displays a check mark.

Another mode, always visible, is relevant neither to the current collection nor to the viewer.

- The current value of the *Working Directory* mode appears in the banner of the program, above the panel, in a field titled Dir:.

State Visible On Direct Request

Some parts of each raw ref. in the collection are only visible when explicitly requested by the user.

- Contents of %X fields (internal comments) only appear in the viewer when Verbose Mode is on.
- The name of the file from which each raw ref. was read only appears (formatted) in the viewer when Sources Mode is on.
- For those refs. that were collected by reading citations from text files, the name of the file from which the citation was obtained only appears (formatted) in the viewer when Sources Mode is on.
- Those parts of each raw ref. that are not formatted at all, %K fields (keywords) for example, are only visible when the user requests it with the operation View Refs. in Raw Format. This information appears in an auxiliary frame.
- External notes that are optionally associated with each raw ref. are only visible when the user requests it with the operation View External Notes on Selected Ref.. This information appears in an auxiliary frame.

One set of mode information, directly relevant to neither collection nor viewer, is likewise not always visible:

- The list of available indexes (with status of each) is only available in a menu that appears when the user right mouse buttons over the Indexes item in the panel, top row at the right. Each available index appears as a choice in the menu; each index that is

currently active has a check mark next to its name in the menu.

Finally, one tangential set of information, is available on request:

- The files of *definitions* used to expand raw refs. are visible when the user requests it with the operation View Definitions. This information appears in an auxiliary frame.

Other State

Some aspects of BiblioText's internal state become visible only indirectly. In particular, arguments are retained between invocations of certain commands. They appear in the input field of typein prompters when a new command is invoked, and the user may reuse them by simply invoking OK. Retained arguments include:

- *keywords* for lookup.
- *input file name* for read.
- *output file name* for saves.

3.4. Command Formation

This section summarizes how the user initiates commands in the current BiblioText user interface. There are two issues, reflected in two separate taxonomies of the available operations: argument discipline and command invocation.

Argument Discipline

This summary lists all functions available to the user, organized around the method for supplying arguments to each.

- *Panel selection or menu.* When a left button press can be used (as for Sort and display modes), command invocation and argument selection are coincident. When a right button press can be used (as for all three), the mouse button down is command invocation, and the mouse moves with mouse button up is argument selection. Functions following this discipline are:

  Sort Collection
  Set Display Modes
  Set Active Indexes

- *Prefix commands.* The command is selected first, then a prompter gathers input arguments. Functions following this discipline are:

  Lookup from Keywords
  Read from File
  Store to File
  Set Working Directory

- *Postfix commands*. The argument (here one or more refs. in the collection) are selected first, then the command is initiated. Functions following this discipline are:

Delete Selected Refs.
Make Citation
View Refs. in Raw Format
View External Notes on Selected Ref.

- *No arguments*. Two commands have no arguments at all:

View Definitions
Delete All Refs.

- *Mixed*. Finally, two standard SunView interface commands are a little bit of each. Some parts of the command arguments are selected coincident with command invocation, others are supplied by mouse movements afterwards:

Scroll Viewer
Resize Frame

Invocation

This summary lists all functions available to the user, organized by location in BiblioText's visual interface where they can be initiated with mouse button press. Each is annotated with the argument discipline identified above.

- *Main Menu*. A right button press over the viewer brings up the Main Menu with the following operations available for selection.

Lookup from Keywords (prefix)
Read from File (prefix)
Store to File (prefix)
Make Citation (postfix)
View External Notes on Selected Ref. (postfix)
View Refs. in Raw Format (postfix)
View Definitions (no args.)
Delete Selected Refs. (postfix)
Delete All Refs." (no args.)
Set Working Directory (prefix)

- *Panel Items*. The top row of the panel displays four items. The leftmost displays the number of refs. and is not active. The next three respond to mouse button presses with command invocations. Note that the two display modes are part of the same item. A

left button press over one of them toggles its state; a right button press over either pops up a a menu with both modes displayed, much like the indexes item.

Sort Collection (menu selection)
Set Display Modes (panel/menu selection)
Set Active Indexes (menu selection)

- *Frame.* Finally, button presses over BiblioText's window boundaries invoke standard SunView functions. Both scrollbar and outer frame responds to a variety of button presses and accelerators.

Scroll Viewer (mixed)
Resize Frame (mixed)

3.5. Thoughts on Model Building

This part of the experiment was my first attempt (after one and one half years of development and use) to craft coherent explanations of BiblioText's behavior. It was surprisingly difficult. This section concludes with some observations on the process of constructing these descriptions.

Separating Functionality from Interface

A surprising source of difficulty was the separation of functionality from interface. This contributed to the failure of the full specification and muddled early attempts at the conceptual model.

For example, one choice of command syntax generated new functional behavior. Commands like Lookup from Keywords are prefix commands, so a prompter appears in which keywords can be typed. The prompter appears displaying a default value; this arrangement requires additional functions like Accept Defaults and Cancel, each of which has its own functional description.

Other examples derive from the inclusion of active SunView objects: panel buttons, menus, scrollbars, and the like. These have their own semantics and functionality, only some of which is configurable by client programs like BiblioText. Use of these objects helps build interfaces, but results in programs for which simple explanatory models are inadequate.

A common counterargument on the matter of interface toolkits is that consistency across applications has its own rewards, and that experienced users will benefit. This advantage can only be gained when the designer can (a) forecast the level of experience in users, and (b) integrate as cleanly as possible the models induced by a toolkit with those models appropriate for the particular application at hand. This can be difficult, and is an instance of the imposition of various contexts on design, to be discussed more fully later (section 4.6 "Thoughts on Theoretical Evaluations").

In the end, I constructed a model that ignores as much of the interface functionality as possible. The model is simpler, but relies heavily on external context for its suitability as an explanation. Users familiar with omitted aspects of the interface will learn to use the program easily; those unfamiliar will find that the conceptual model doesn't help them at all. For example, a user unfamiliar with SunView might be unable to locate frame operations, and might never think to get the main menu with a right mouse button press over the window's narrow boundary. The popup prompter, on the other hand, might be a bit unfamiliar to the user who has never seen a Macintosh in operation.

Task vs. Conceptual Models

My first attempts at explanation derived from an earlier functional design, but contained much more detail. All attempts to organize the volume of detail failed to produce a coherent whole, however. One explanation might be that both my original design and intervening use had always been implicitly oriented around task descriptions and had been much more like Young's simple model of the ALG calculator [You81] than like a conceptual or predictive model. The confusion may be an instance of what diSessa calls the *hacker bug*, "in the extreme, to allow a structure for each function in the language" [diS86].

As further evidence, I found it necessary to invent concepts that had never been part of my understanding of the program. For example, even though I had not thought of them that way, it became necessary to describe the *collection* and the *viewer* as separate conceptual objects, each surrounded by its own functionality. This helped explain troublesome parts of BiblioText's behavior such as:

- The user can edit (in a sense) the collection, but not the contents of a particular ref.
- There are many visual presentations possible for each ref. Differences among representations are confounded a bit by *abbreviations* and various display modes.

I never did manage to explain some things conceptually. For example, the discrepancy between textual selection (as highlighted in the viewer) and the current *selected refs.* on which some commands operate derived from an implementation restriction and admitted to no other coherent explanation.

Categorizing The Sort Function

The attempt to clarify a conceptual model turned up an interesting question. Should the Sort function be considered an operation on the collection (like adding and deleting refs.) or a viewer option (like setting viewer modes)? It has overtones of each. For example, the current sort order persists somewhat and had always seemed naturally situated on the panel along with two other items that are both *modes*; on the other hand, all other operations that edit the collection seem naturally situated in the main menu. This suggests Sort as a viewing mode. And yet sorting seems intuitively like doing something to the object being sorted.

The conceptual distinction depends on how one defines the collection. If the collection is defined to be an ordered *list*, then Sort is an operation on the collection since it changes it. If, on the other hand, the collection is defined to be a *set*, then Sort does not change it and is merely a viewing option (like Verbose Mode).

I originally concluded that this distinction is an artifact of attempts to impose a needlessly detailed model onto BiblioText, and that it is of no consequence to users. Sorting is reasonably well understood, BiblioText's response to Sort operations is predictable, and it would be hard to explain the distinction to a user.

There turned out to be more to it, however, as later analysis revealed (section 5.4 "The Editing Model"). Attempts to design the semantics (a conceptual model) for a general *undo* function in BiblioText made the distinction between the two interpretations crucial. If Sort is an operation (like other edits), then a sensible and intuitively understandable conceptual model for *undo* can be designed. The necessary model of *undo*, by the way, is much like Young's simple verbal model of the ALG calculator; here it is simply "reverse the last operation performed." If, on the other hand, Sort is a viewing mode, then a model for *undo* can be designed, but it admits to no comprehensible explanation.

Minor bugs in model

Creating the models uncovered several places where functionality is inherited from implementation and cannot be reasonably explained. These issues had never appeared (or didn't seem to matter) when taking the task based approach to BiblioText's design.

For example, each ref. has associated with it the name of the file from which the ref. was obtained, and may have associated with it the name of a file from which a citation was read that identified the ref. However, refs. can be added from multiple sources. During a single lookup, a ref. might exist in more than one index. During successive lookups, acquired refs. are sometimes not added if they would duplicate a ref. already present (even if located in a different index). The same applies to citations, since a ref. might be cited multiply in one file or in several files. In practice, file names become associated with each ref. at the time it is first acquired, but this doesn't always have a clear analogue in the conceptual model.

4. Theoretical Evaluations

This section continues the first phase of the experiment, an evaluation of BiblioText as it existed at the beginning of the project. This evaluation describes version 3.3, referred to as *the old interface*; figures 3.1 and 3.2 in the previous section show the old interface in operation.

The evaluation began with a set of descriptive models, each focusing on different aspects of the program (section 3 "Descriptive Models"). This section moves the evaluation to a higher level of abstraction by by judging the old interface from different points of view, many of them theoretically motivated.

Each part of this section adopts a particular point of view, and notes the motivation for each. Several of these amount to an attempt to follow the advice implied by selected works from the research literature on human interface design. These approaches are necessarily informal, because the literature is seldom prescriptive in any useful sense.

When the evaluation leads to suggestions for improvement, these appear in specially typeset paragraphs. The collected suggestions from this evaluation are the raw material for the next phase of the experiment (section 5 "Redesign").

A recurring difficulty during the work reported in this section was the need to place BiblioText in some particular context before useful judgements could be made. The concluding part of this section reports on this difficulty in more detail, along with thoughts on the relevance of the various theoretical points of view.

4.1. Intuition

The first point of view for judging the interface is the one that had been in effect all along: intuition based on personal experience. A few points were on my list of things about the interface that "weren't quite right."

- Each panel item is configured to behave a bit differently than the rest.
- The user selects characters in the viewer, whereas BiblioText commands operate at the granularity of whole refs.
- The main menu is cluttered, disorganized, and slow to use.

These arise again in the evaluations below, along with suggestions for their repair.

4.2. Task Analysis

The next point of view for judging the interface returns to the motivation for building the original prototype of BiblioText, an inventory of the user tasks the program is intended to support. This section reconsiders each of those tasks, describing step by step how a user might accomplish the task using BiblioText.

This review should yield a sense of the task domain and how it must be mapped into the domain of BiblioText. In the terms of Don Norman, this should reveal a bit about the *gulf of execution* [Nor86], both *semantic* and *articulatory distance* that must be spanned [HHN86]

Locate a Ref.

Several tasks begin with the identification of some specific document, and in particular to find the ref. that acts as a surrogate for the document of interest. The purpose of the *Locate a Ref.* task is to get this surrogate into the current collection and visible to the user.

Thus, this task provides the context for many other tasks. However, it is inherent to browsing that this task is in turn the result of an indeterminate collection of other tasks: keyword lookup, scrolling, sorting, and the like. The *Locate a Ref.* task appears explicitly, however, to serve as a reminder that various tasks in this interactive environment are interconnected in ways that defy simple analysis.

Locate Notes on a Specific Document

The first step establishes the context, namely that some interesting ref. is in view as a surrogate for the document.

(1) Locate ref. by some context dependent means, using keyword lookups, file loads, scrolling, sorting, and the like.

In the current arrangement, notes can be stored in two ways. They can be in an external file named in a %Z field, or they can be internal to the raw ref. data, in a %X field. We'll

look at the former case first.

(2) See if external notes are present. If so, [notes] appears in the formatted output.

(3) Select ref. with a left button press over its display.

(4) Request notes with view NOTES in the main menu.

In the second case, internal notes are already on display if Verbose Mode is on. If not:

(2) See if internal notes are present. If so, [X] appears in the formatted output.

(3) Turn on Verbose Mode with a left button press over the panel item.

The last step here has the effect of making the entire collection displayed verbosely, a bit more than what was wanted.

SUGGESTION: Allow Verbose Mode to be set independently for each ref.

Locate a Specific Document

The task here is to discover where a document is physically stored, once a ref. describing it is located (as above) in some other task context.

(1) Locate ref. by some context dependent means, using keyword lookups, file loads, scrolling, sorting, and the like.

Information about the physical whereabouts of a document can be recorded in two ways; both may be necessary. The first method uses the name of the file from which the ref. was found. In the presence of multiple indexes (belonging to multiple people) this points to the person whose database contained the ref. Individuals may use the file name to further specify physical location within their own databases.

(2) Turn on Sources Mode, if not already on, with a left button press over the panel item. This may not help users who have only one index and whose refs. reside in an undifferentiated mass.

The second way to record location data is in a %W field, included in the *bib* raw data specification, but without further guidance or constraints. One might store in these fields the owner's name (when it belongs somebody else), a call number (when obtained from a library), or a page number (when a reprint appears in a tutorial or other bound collection of reprints).

(3) See if internal location data is present. If so, [W] appears in the formatted output.

(4) Select ref. with a left button press over its display.

(5) Request raw display of selected ref. with view RAW references in the main menu.

This seems a bit awkward when compared to the handling of internal notes in %X fields. All that is wanted here is a single field of information, usually only a word or two in length.

SUGGESTION: Display [W] <locn.> when Verbose Mode, is on, analogous to the handling of [X].

This leads in turn to the suggestion made above, that the choice of Terse/Verbose Mode be made on a per ref. basis.

Write a Document, Citing Other Documents

This task presumes that an editor somewhere on the screen is currently active and contains the document being written. And, as above, it presumes that some other task has resulted in the location of the ref. describing the document to cite.

(1) Locate ref. by some context dependent means, using keyword lookups, file loads, scrolling, sorting, and the like.

(2) Select ref. with a left button press over its display.

(3) Request citation with make CITATION in the main menu.

(4) Insert citation into document using some variant of the SunView interwindow copy operation.

Maintain for Later Retrieval a Collection of Related Documents

The browser does not support, nor does it suggest a specific way to accomplish this task. One possibility is to keep the collection as a separate file of refs. in raw format.

(1) Get old collection for perusal and maintenance, using LOAD from file in the main menu.

At this point, one might want to make a variety of editing changes. Those supported are adding, deleting, and sorting, but these are insufficient for generalized editing.

> SUGGESTION: Complete basic editing functionality to allow arbitrary rearrangement of the collection.

The task of adding new refs. can be done two ways. The first, already described, is by reading from another file of raw refs. The second is a keyword search, discussed below under the task "Extract a New Group of Documents ...". Deleting refs. involves the following steps.

(2) Select refs. to be deleted with a left button press over the display of the first ref. and, if more than one is to be deleted, a right button press over the display of the last.

(3) Request deletion using DELETE refs. in the main menu.

It may be desirable to maintain the collection in some order. The browser can sort according to several standard criteria.

(4) Request a sort by selection from the right button menu over the Sort panel item.

Finally, one can save the collection for later reference.

(5) Store collection using STORE to file in the main menu.

This general approach is deficient in several ways, mainly with respect to data representation. First, such a collection redundantly stores copies of refs. copied out of a database, and is therefore inefficient and insensitive to updates to the original data. Second, there is no convenient way to integrate notes on the maintained collection with its storage. These notes are crucial; insofar as one of these collections is a surrogate *pile* of reading material organized by subject or project, additional notes must be made to play the role of written marginal comments, paper clips marking pages, and Post-it[9] notes suggesting

[9] Post-it is a Trademark of 3M Inc.

what part of the document is relevant to the subject or project at hand.

A workable solution, is to write a separate document that is, in effect, an annotated bibliography. It requires slightly abnormal batch processing by *bib* and *troff* to print, and only calls upon BiblioText for two tasks. The first task is to locate new documents and prepare citations, discussed earlier. The second task is to read a current version of the bibliography and collect the refs. for all its citations.

(6) Read an old collection using LOAD from File from the main menu.

There should be better solutions.

> SUGGESTION: Create and support some specialized storage format(s) that can be read, maintained, and stored directly by BiblioText; make the storage external to the *bib* database so no indexes are affected, and refer to documents in the database with imprecise citations.

Extract a New Group of Documents According to Unanticipated Criteria

This is the most exploratory, unstructured task. The heart of the task is the keyword search.

(1) Request a keyword search using LOOKUP keywords in the main menu.

(2) Enter keywords into the typein prompter that appears.

(3) Start search with a left button press on OK in the prompter, or with an accelerator, the RETURN key.

(4) Scan resulting collection using scrolling.

For some purposes, scanning might be aided by sorting or by an iterative combination of deletions and sorts, both discussed already. A subsequent keyword search positions the viewer at the first ref. added; this is to the first *new* ref. found by the search, since duplicates are screened. BiblioText displays statistics (number of refs. found, number of refs. added) that help the user evaluate the outcome of each search.

Typing strings into prompters can be redundant and tedious, especially when the words to be typed have already appeared on the screen. For example, new keywords are often suggested by the results of prior searching.

> SUGGESTION: Support keyword lookup without requiring that interesting, visible keywords be retyped into the popup prompter.

Likewise, searches might lead one to discover, when Sources Mode is on, a new file of raw refs. that appears of special relevance.

> SUGGESTION: Support file loading without requiring that visible file names be retyped into the popup prompter

Peruse Personal Notes, Either from a Maintained Collection or Following Some Unanticipated Criteria

The task here can start with a maintained collection or following some unanticipated criteria. This calls for iterations of tasks already discussed: locating refs., viewing notes, and making new collections. One can imagine wanting to ''spread out'' notes from several sources, but BiblioText only supports the viewing/editing of one external file of notes at a time.

SUGGESTION: Allow multiple files of external notes to be viewed/edited simultaneously.

Preview Citations in a *troff* Document

This task, and the one following, did not appear in the original task analysis, since that analysis presumed no particular automatic tools. These two new tasks emerge only in the presence of BiblioText and related software.

The original task analysis presumed nothing about *troff* or the imprecise citations associated with *bib*. Given this mechanism, however, it is useful to quickly preview the refs. being cited in a document, without waiting for batch processing by *bib* and associated phototypesetting. The preview help in two ways. First, it previews the references section of the document, allowing the user to see what is being cited (and not cited). Second, checks the correctness of the embedded citations; that these can be incorrect is a known defect of the *bib* representation scheme.

(1) Empty current collection, using DELETE ALL in the main menu.

This first step ensures that the refs. in the collection are only those cited in the document of interest. It is not necessary to empty the collection if the task is only to check for errors.

(2) Lookup citations in document, using LOAD from file in the main menu.

If an auxiliary frame titled errors in file <file name> appears, then the file contains citation errors. The auxiliary frame describes errors, one per line: source line number, keyword string, and error type. The user can edit the document and use BiblioText to generate correct citations as replacements.

Maintain *bib* Data

Given the design constraints of BiblioText, it can offer almost no support for the modification and maintenance of *bib* raw data and indexes. One exception, however, is to display for easy browsing the files containing definitions and their expansions. This simplifies the job of entering new *bib* data.

4.3. Metaphor

Opinions vary on the value of metaphor in the design user interfaces. Some writers are enthusiastic and prescriptive [CaT82], and others are cautionary [HaM82]. In any case, a certain amount of this needs to be discussed, since every user will carry around experience with artifacts that are like BiblioText in certain ways.

The search for natural metaphors yielded three different answers to the question: "what is BiblioText like?" Each answer naturally suggests making BiblioText more like the object of comparison. Conversely, each case can warn that differences between BiblioText the object of comparison might be misleading to a user.

The Editor Metaphor

BiblioText is something like *an editor*, since it displays the contents of a buffer (the collection) and supports operations like adding, deleting, and saving.

It differs from editors in two crucial ways. First, it is specialized to operate at the granularity of refs., not characters. Since users with editor experience will find this unfamiliar, the interface should make it abundantly clear. Unfortunately, selections are currently highlighted at the granularity of characters, not refs., exacerbating this misleading part of the metaphor.

SUGGESTION: Alter the feedback response to selection so that highlighting occurs only at the granularity of refs.

The second way in which the editor metaphor fails is in missing functionality. Coarse granularity notwithstanding, a user might well expect to be able to move refs. about arbitrarily within the collection. BiblioText only supports two kinds of edits: additions at the end of the collections and arbitrary deletions. There is no technical reason for not supporting more general editing functionality; this analysis suggests that adding functionality to make BiblioText more like familiar editors might make it simpler to understand.

SUGGESTION: Complete basic editing functionality to allow arbitrary rearrangement of the collection.

The List Metaphor

BiblioText displays *a numbered list of refs.* This is like numbered lists in general, and particularly like printed bibliographies. Most numbered lists, however are static; they are hard to rearrange and renumber quickly.

One way to make the display more like familiar printed bibliographies would be to add elaborate (and selectable) formatting. But this corresponds precisely to a piece of the original functional design that was eventually scrapped. Fancy formatting serves a role in printed bibliographies, but tasks using the browser require quick skimming and uniformity.

Another way to imitate printed bibliographies would be to allow annotation. This is only weakly supported at present, since annotations must either reside in raw ref. files (and therefore immune to change by BiblioText) or in external files (and therefore require special purpose, offline processing to produce the annotated printed bibliography in a familiar form). This leads to a suggestion that appeared earlier.

SUGGESTION: Create and support some specialized storage format(s) that can be read, maintained, and stored directly by BiblioText; make the storage external to the *bib* database so no indexes are affected, and refer to documents in the database with imprecise citations.

The Note Card Metaphor

BiblioText supports the collection of references and notes in much the same way that many people do with *note cards*.

BiblioText is specialized, however, and therefore not very good at many of the tasks for which cards are well suited. The program NoteCards is a much closer approximation to ordinary note cards and was inspired in exactly this way, but it has the complementary weakness: "many users have found the task of structuring and processing their ideas in NoteCards to be relatively difficult" [HMT87]. In contrast to NoteCards (and to the

more recent Hypercard [Goo87]), BiblioText supplies structure that is both suited to the specific tasks at hand and optimized for them (consider the chore of skimming 100 refs. stored one per card, looking for titles relevant to some subject).

4.4. Direct Manipulation

This section considers BiblioText in light of discussion about *direct manipulation* interfaces, and with particular reference to the analysis presented by Hutchins, Hollan and Norman [HHN86]. That paper begins with a summary of Schneiderman's three criteria, and the first evaluation of BiblioText proceeds from them.

Schneiderman's Criteria

Criterion 1. *Continuous representation of the object of interest.*

This is generally true of BiblioText, since the collection is always on display in the viewer. It is limited somewhat by the necessity for scrolling, since the total amount of information in the collection can be large.

A related approach is to review the amount of BiblioText's internal state that is *not* continuously visible (section 3.3 "State and Visibility"). Much of the state in this category is of little ongoing interest, such as default arguments being retained between commands. Other state in this category is not interesting because of size; it would present too much information for quick browsing (such as the list of available indexes or all external notes available for the current collection).

Some information contained in (or associated with) raw refs. is only visible in the viewer by special request. The user controls the display by switching Verbose Mode and Sources Mode. In some cases, information is merely elided when not present (internal comments in %X fields), but in others it simply disappears (file names for Sources Mode). Finally, there is some information in raw refs. that is never formatted (keywords in %K fields); to see it the user must select the ref. and request a separate display in raw format. This arrangement seems a bit arbitrary.

> SUGGESTION: Generalize the method for formatting raw refs. When Verbose Mode is off, attempt to display *some* representation of everything present, elided in many cases. When Verbose Mode is on, display as much more as possible, consistent with the goal of convenient browsing.

Criterion 2. *Physical actions or labeled button presses instead of complex syntax.*

This is generally true of BiblioText, since all commands are initiated with button presses. However, as summarized earlier (section 3.4 "Command Formation") the complete specification of commands requires various prefix and postfix methods for argument identification.

> SUGGESTION: Make command formation either consistently prefix or postfix.

Further, the main menu is large, disorganized, and provides little clue to general kinds of functionality of commands.

> SUGGESTION: Rationalize the main menu to make it easier to locate desired operations.

Criterion 3. *Rapid incremental reversible operations whose impact on the object of interest is immediately visible.*

Most operations in BiblioText are *incremental* and their effect on the object of interest, the collection, is *visible*. A rationalized main menu (suggested above) should make operations more *rapidly* initiated; and most operations (on a Sun-3) execute quickly. Operations that execute slowly should make announcements to that effect during their operations, but at present only one or two of them make such an announcement.

SUGGESTION: React quickly in some way to every command invocation. Make clear to the user when the program is working and not available to respond to other input.

Few BiblioText operations are reversible. Hutchins, Hollan and Norman [HHN86] disagree about the absolute importance of reversibility, but it might be helpful for BiblioText.

SUGGESTION: Make all operations on the collection reversible.

Criteria for Engagement

Hutchins, Hollan and Norman [HHN86] discuss aspects of direct manipulation interfaces in terms of *distance* and *engagement*. The task-based evaluation above approaches the notion of directness by walking through typical tasks in the task domain. This section evaluates BiblioText in terms of their four minimal requirements (all subjective) for the feeling of direct engagement.

Criterion 1. *Execution and evaluation should exhibit both semantic and articulatory directness.*

In the extreme approach to this, typified by NoteCards and Hypercard, each card (cluster of text) is separately manipulable. Operations on individual refs. in BiblioText are less direct, but for reasons already described (section 4.3 "Metaphor") more directness would result in less convenience for its most important tasks. BiblioText is specialized and can easily perform some indirect operations (such as Sort) that would be difficult to achieve directly. This demonstrates a point Hutchins, Hollan and Norman make in their section on Problems with Direct Manipulations:

> It is important not to equate directness with ease of use. Indeed, if the interface is really invisible, then the difficulties within the task domain get transferred directly into difficulties for the user.

Criterion 2. *Input and output languages of the interface should be inter-referential, allowing an input expression to incorporate or make use of a previous output expression. ....*

BiblioText is inconsistent here. Refs. can be selected easily as arguments for operations, but other interesting objects cannot. Two prior suggestions (section 4.2 "Task Analysis") suggest that visible entities other than refs. be easily selected as command arguments.

SUGGESTION: Support keyword lookup without requiring that interesting, visible keywords be retyped into the popup prompter.

SUGGESTION: Support file loading without requiring that visible file names be retyped into the popup prompter.

This could be generalized even more. For example, the string [notes] in the viewer is a surrogate for external notes, but requests to see those notes must now be initiated by selecting the ref. and then invoking a separate command from the main menu.

SUGGESTION: Invoke the command view NOTES by buttoning the display of [notes] in the viewer, rather than by selection from the main menu.

Criterion 3. *The system should be responsive, with no delays between execution and the results, ...*

This issue appeared already in the discussion of Schneiderman's criteria, along with a suggestion for improvement.

SUGGESTION: React quickly in some way to every command invocation. Make clear to the user when the program is working and not available to respond to other input.

Criterion 4. *The interface should be unobtrusive, not interfering or intruding....*

This is subjective and defies analysis by the designer. Empirical evidence from users may help with an evaluation here.

4.5. Visual Consistency

Several problems with visual consistency appeared, a side effect of the long and thoughtful look at each piece of the BiblioText display occasioned by the various evaluations. These issues arose independently of any particular theoretical motivation, although the need for visual consistency is implicit in most of the cited literature.

Active vs. Nonactive Panel Items

Between four and six items appear on the panel (depending on how one counts). Some of them respond to mouse clicks (the Sort item, the two display modes, and the Indexes item), one does not (the current ref. count), and one responds only to selections (the message line). There are no reliable visual clues to suggest which might respond and which might not.

SUGGESTION: Provide visual cues to distinguish active from nonactive panel items.

Active Item Display and Behavior

Each of the three active panel items appears and acts differently; this results from trying to closely match each item's unique semantics to a suitable SunView panel item, without considering its neighbors in the panel. Semantically, the Sort item allows one choice from many, the Display Modes item contains two binary switches, and the Indexes item contains a variable number of binary switches. Behaviorally, the Sort item always displays its current setting, the Display Modes item appears as two labeled boxes that can be left buttoned separately, and the Indexes item appears simply as Indexes; all three can be right buttoned for a menu. Taken as a whole, variations among three items are potentially confusing.

SUGGESTION: Make the appearance and behavior of the three active panel items as much alike as their semantics, and SunView support, will allow.

Current Working Directory

According to the conceptual model the *Current Working Directory* (cwd) is a mode; its current value affects the operation of some commands. There are two issues surrounding the cwd: where to display its current value, and how the user initiates the command to change it.

The cwd's value now appears in the banner, at the top of the main frame. One advantage is compactness, since space in the banner is marginally free. It is also conceptually compatible with general UNIX functionality; cwd is a background property of most programs and the banner already suggests a boundary between application and system (it almost always displays the name of the application, for example). Finally, this display technique for cwd appears (exactly as in BiblioText) in an example supplied in the "Sun User Interface Conventions" appendix to the SunView manual [SSS86].

On the other hand, cwd is one of three modes, and the other two (display modes and active indexes) appear in the panel (though their values are not always visible). This view suggests that the three modes appear together, since they are functionally related in this respect.

SUGGESTION: Display the value of Current Working Directly alongside the other modes in the panel.

The second issue concerns command initiation. Cwd is changed by selecting set Working Dir. from the main menu (and then typing into a popup prompter). However, the main menu is associated with the display of the collection, conceptually distant from the cwd; this would make it hard to locate. And as a side effect, the command adds to the existing clutter in the main menu.

SUGGESTION: Move initiation of set Working Dir. from the main menu to a location more consistent with its conceptual and visible position.

Auxiliary Frame Labels

Four different auxiliary frames appear at various times during BiblioText's operation: raw data display, external notes display/editor, bib definitions display, and error messages associated with reading citations from a text file. Each has a small panel at the top, but each has a different style of identifying label. Some do not explicitly announce that the auxiliary frame is associated with BiblioText.

SUGGESTION: Label auxiliary frames consistently: a BiblioText label, and some message describing particular contents.

Auxiliary Frame "Done" Operations

Every auxiliary frame has the Done operation (make the frame disappear) available in its standard SunView frame menu. BiblioText's auxiliary frames make this operation available redundantly in other ways. Three of the frames have an entry labeled Done in the main menu associated with the viewing area; this approach involves pushing the standard text subwindow (editing) menu into a submenu labeled textsw=>. The bib

definitions frame, on the other hand, has a panel button item marked Done in the panel; its text subwindow menu is unchanged.

Aside from inconsistency, this arrangement makes standard SunView editing functions inconvenient. This is moderately important for the read-only frames (where search is the only SunView command likely to be heavily used), but the viewer for external notes may also be used to edit.

SUGGESTION: Standardize Done command invocation across all auxiliary frames.

4.6. Thoughts on Theoretical Evaluations

Although the value of these evaluations cannot be judged until the remaining phases of the experiment are complete, several problems became clear during this phase. Comments on some of these follow.

Problems with Metaphor

Carroll and Thomas offer numerous suggestions for the use of metaphor in computing systems [CaT82], but their discussion is irrelevant to BiblioText and this experiment. They consider how naive users learn a system and eventually become experienced. Here, beginners are anything but naive; they are subject to many other influences, especially opportunities to draw comparisons with other programs (including *bib* and *lookup*). Instead of comparing BiblioText to non-computer artifacts, we need to think about how BiblioText is like the SunView text editor, like *emacs*, or like the MacIntosh *clipboard.*

Metaphor is important, and three were identified in this evaluation (editor, list, and note card). None of the metaphors is adequate alone. In combination, however, (in particular the list and note card metaphors) they appear to have great explanatory power. This patchwork is an example of the *distributed models* discussed by diSessa [diS86].

Problems with Direct Manipulation

One problem with direct manipulation was mentioned in the evaluation, a problem identified by Hutchins, Hollan and Norman. BiblioText deliberately sacrifices directness to make some operations in the task domain (such as sorting) easier for the user than if direct manipulations were required.

A more general problem with this type of evaluation is that it depends heavily on the task domain and the way one views it. In practice, one can make observations about the task domain, but they are subject to influences difficult to analyze. For example, variations in working environment can dramatically affect how an individual user conceptualizes the task domain. Furthermore, the presence of a new tool like BiblioText itself affects how a user will think about the task domain. Finally, analysis of how a person conceptualizes the world is sketchy under the best of circumstances.

The Problem of Context

As the preceding comments make clear, the evaluations during this phase of the experiment were repeatedly confounded by the need to place BiblioText in its working environment. Many judgments depended on an understanding of the *assumptions*

prevalent in that working environment.

The design of BiblioText was influenced by many design and implementation paradigms external to the project, a few of which appear in the list below. Their variety and unpredictability suggests the difficulty of creating conceptually clean designs in existing contexts.

- *UNIX Conventions.* BiblioText users are, by definition, experienced with UNIX. For example, SunView imports UNIX typing protocols that affect the behavior of BiblioText's popup prompters: DEL erases the previous character and Control-U erases the whole line. Another important example is the file system based on path names; files can be identified explicitly or by typing part of the path name to which is prepended a bit of contextual state, the *current working directory.*
- *Bib Data.* Users are familiar with the *bib* tools at some level. They already understand raw format, imprecise citations and keywords, indexes, and the like. They may not be aware of all possible fields, though, and they probably don't appreciate the fine points of erroneous multiple matches in the database.
- *SunView Support.* A major constraint on the design of the interface is a simple one. SunView provides high level support, making possible interactive, visually oriented tools like BiblioText. At the same time, that approach limits design options to those which are supported; many styles of interaction are not supported.
- *SunView Expertise.* Users have some level of functional competence with standard SunView operations. These include: basic operation with the cursor, frames, menus, buttons, and the like; the selection service for moving text among windows; and the SunView (textsw) editor.
- *Editing Paradigms.* Most users understand a variety of editing paradigms. Those in the local working environment rely on either *vi* and *emacs* (which, by the way, differ significantly), but many also use microcomputers that support simple cut/paste editing. Macintosh users know what a *clipboard* is and how to use it. On the other hand, nobody in this environment uses the SunView editor that constitutes BiblioText's viewer.
- *Changing Software Context.* Contextual influences are in continual change. During BiblioText's evolution prior to this experiment, enormous changes were wrought by successive releases of SunView. Not only do new components with newly designed functionality appear (panel items, text subwindows in the form of editors, etc.), but conventions for their use change. This is explicit in the "SunView Interface Conventions" appendix in the current manual [SSS86], where comments appear like "The best use of the middle button is still being discussed." and elsewhere "... is *not* recommended, as it conflicts with future plans ... ."
- *Changing Hardware Performance.* A bit more subtly, changing workstation performance has an effect. Operations that once ran slowly, and therefore required some feedback during the interim, are now nearly instantaneous running on a Sun-3.
- *Changing User Expertise.* Users themselves change with exposure to new tools and new paradigms of interaction. For example, the term *clipboard* made popular by the MacIntosh computer is more widely understood than it was one and one half years earlier, when the first prototype of BiblioText was built. SunView implements (with release 3.0) an elaborate extension of the *clipboard* model for interwindow operations

that makes use of mouse selection and specially marked function keys (among others, PUT, GET, and DELETE). Unfortunately, local experience suggests that most users at present understand only small fragments of this complex model. This presents difficult design choices: either mimic the entire SunView editing model for complete integration, and risk making BiblioText as incomprehensible as SunView editing, or detach from the SunView model entirely and invent yet another editing paradigm for users who already have to cope with many different paradigms.

5. Redesign

This section presents the second phase of the experiment, a rework of BiblioText's user interface based on results from the first phase: models (section 3 "Descriptive Models") and evaluations (section 4 "Theoretical Evaluations"). The modifications described here produced version 3.4, referred to as *the new interface*; figures 5.1 and 5.2 below show the new interface in operation.

Each part of this section begins by recapitulating a group of the suggestions that arose during evaluation of the old interface. Each group corresponds to some general area of BiblioText's interface (and in one case an area of functionality), so that overlapping and conflicting suggestions can be resolved together.

The discussion results in a design decision for each suggestion. Some suggestions are elaborated into specific design changes. Other suggestions are rejected, either because they conflict with other suggestions and high level goals, or because they are technically infeasible. A few suggestions are postponed because of time limitations for the project.

One recurring theme of this section (foreshadowed in the previous section) is complex interdependency. Thoughts on the problems this causes appear in the final part of this section.

5.1. Cosmetics and Visual Consistency

This group of suggestions addresses general visual improvements to BiblioText's old user interface.

Active vs. Nonactive Panel Items

SUGGESTION: Provide visual cues to distinguish active from nonactive panel items. *Decision: Use different fonts and separate lines.*

The new interface presents two visual cues that reveal this distinction. The three active items appear in the first line, and they appear in a bold font. The two informative items (number of refs. in collection and informative messages) appear in the second line in the standard font.

Active Panel Item Consistency

SUGGESTION: Make the appearance and behavior of the three active panel items as much alike as their semantics, and SunView support, will allow. *Decision: All operate with menus; none display current values.*

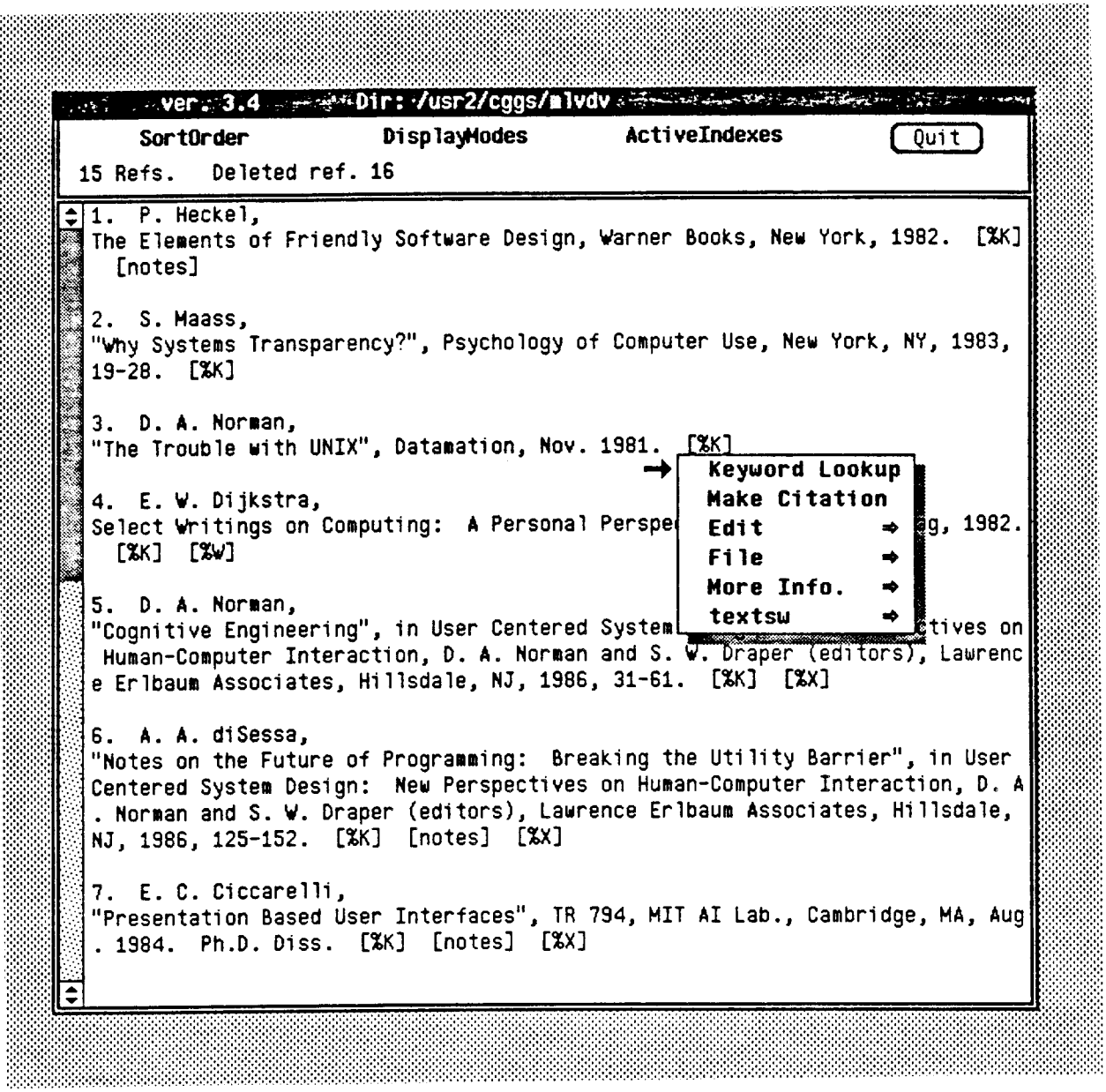

Figure 5.1
The new interface: main menu visible

It would be difficult and wasteful of screen space for Indexes to display its settings continually, like the other two panel items. It doesn't seem imperative to continually display current values of the other two items either, so reconfigure the other two to appear more like Indexes. In the new interface, all three appear as only a (bold) text label, renamed slightly to SortOrder, DisplayModes, and ActiveIndexes.

The three behave much more alike than in the old interface. Each responds to a right button press with a menu that contains choices with checks; the remaining difference is that only one choice in the SortOrder menu can be selected at a time, whereas the other two menus contain multiple binary switches. This final distinction, required by the item's differing semantics, manifests itself in one unfortunate way. SunView *choice items*, like SortOrder, respond to an *accelerator*, a left button press that advances the choice by one with no feedback other than the current value changing (if being displayed,

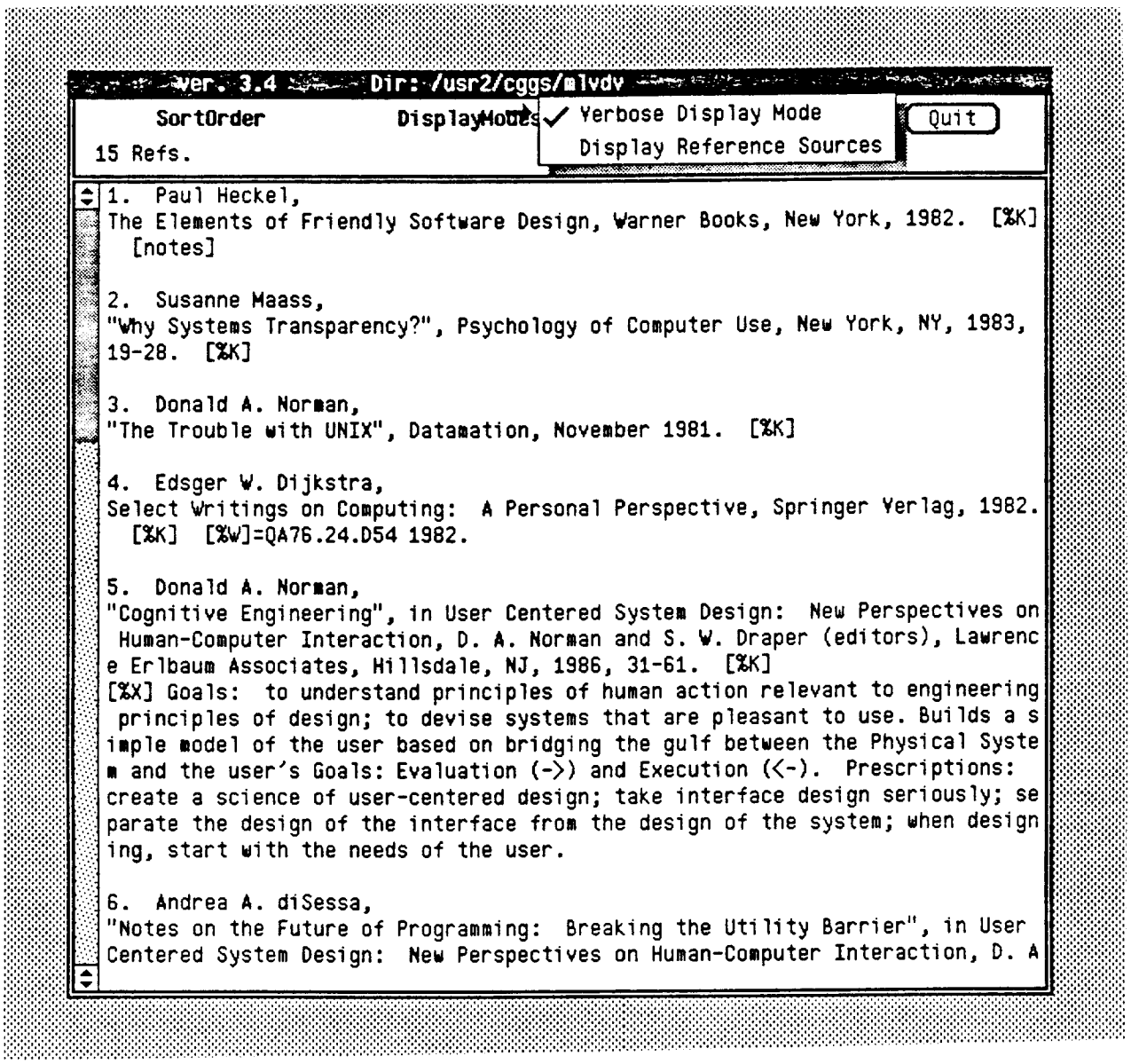

Figure 5.2
The new interface: display mode menu visible

which for SortOrder is no longer the case). Thus, SortOrder is sensitive to left button presses and the other two items are not. It is apparently not possible to disable this feature.[10]

Current Working Directory

SUGGESTION: Display the value of Current Working Directly alongside the other modes in the panel. *Decision: Reject.*

[10] It became clear later how to disable this feature, through use of some advanced SunView programming features.

SUGGESTION: Move initiation of set Working Dir. from the main menu to a location more consistent with its conceptual and visible position. *Decision: Move command to frame menu.*

These suggestions arose separately during evaluation, but must be resolved together. The "principle of engagement" encourages a strong relationship between the initiation of a command and its visual feedback.

The first suggestion is rejected. Arguments in favor of the old arrangement, presented earlier (section 4.5 "Visual Consistency"), are more convincing. Even though Current Working Directory is conceptually a mode in BiblioText, along with display modes and indexes, it is a concept so firmly rooted in the world of UNIX, so closely associated with the boundary between program and operating system, that it demands to be treated specially. Its presence in the panel with other application specific information would not be helpful.

Given the above decision, there seems to be no completely satisfactory response to the second suggestion. The best appears to be moving the set Working Dir. command to the *frame menu*, a general technique discussed in the SunView manual. In the new interface, one sees this entry by making a right button press over the frame; the result is a menu with two entries: Frame=> and Set Working Directory. The first entry announces a submenu that contains standard frame manipulation commands, such as close and Quit.

This solution has the drawback that experienced SunView users may have trouble finding it at first, since this approach, even though suggested in the SunView manual, appears in none of the standard SunView programs or in any application programs in our environment.

Auxiliary Frame Labels

SUGGESTION: Label auxiliary frames consistently: a BiblioText label, and some message describing particular contents. *Decision: Standardize labels.*

In the new interface, each auxiliary frame has a label that begins with BiblioText:. The rest of the label describes the contents of the particular frame, for example raw ref. data, or errors in file <filename>.

Secondary Frame *DONE* Buttons and Menus

SUGGESTION: Standardize Done command invocation across all auxiliary frames. *Decision: Use a* Done *panel button in every auxiliary frame.*

For consistency, each auxiliary frame in the new interface has a panel button item labeled Done, positioned at the right. This is preferable to adding it to the viewer menu for several reasons. First, a button item in the panel seems conceptually closer to the frame than to the text display. Second, it costs no extra size in the frame. Finally, this leaves the standard text subwindow menu in its customary configuration for those cases where the user is allowed to edit the data on display (notably external notes).

An additional piece of consistency is gained in the new interface by adding a similar panel button labeled Quit to the panel of BiblioText's main frame. This is redundant, since it duplicates the equivalent operation in the standard SunView frame menu, and many application programs do not have such a button. However, the change to the frame

menu discussed above (Current Working Directory) pushes the Quit command down one level into a less convenient submenu. All other entries in the frame (sub)menu have redundant keyboard accelerators that allow experienced users to bypass the menu, but Quit does not. In combination, these two factors argue for the addition of a Quit button to the panel.

Signal When Working

SUGGESTION: React quickly in some way to every command invocation. Make clear to the user when the program is working and not available to respond to other input. *Decision: Display a special message during all computations.*

All commands have visual effect, but some are subject to processing delays. In the old interface the message

sorting, please wait ...

appeared while BiblioText was sorting, since this is likely to be time consuming, but no analogous message appeared for other operations. The new interface includes a generic mechanism for announcing the beginning and completion of any non-trivial internal operation; it announces

<doing whatever>, please wait ...

when it begins, and clears the announcement when finished.

It would also help to change the cursor to an hourglass during these intervals; this is also becoming more commonplace, and users can be expected to know what it means. Unfortunately, SunView does not support cursor changes in text subwindows.[11]

5.2. Command Formation

A complex of related suggestions arose for the improvement of command syntax; these include argument selection, prefix vs. postfix models, menu arrangement, and accelerators. Most of them need to be treated together.

Interoperability and Command Syntax Consistency

SUGGESTION: Support keyword lookup without requiring that interesting, visible keywords be retyped into the popup prompter. *Decision: If available, use SunView global text selection as argument.*

SUGGESTION: Support file loading without requiring that visible file names be retyped into the popup prompter. *Decision: If available, use SunView global text selection as argument.*

[11] Cursor changes in text subwindows have since become possible.

SUGGESTION: Make command formation either consistently prefix or postfix. *Decision: Commands are postfix, with exceptions.*

These suggestions argue for an overhaul of the basic syntactic model of command formation; they interact with one another and with the availability of SunView support mechanisms.

The model from the old interface, as summarized earlier (section 3.4 "Command Formation"), is disjoint. Commands that operate on refs. in the collection are postfix: the user selects refs. and then invokes the commands. Commands that require string arguments, such as keyword lookup and file operations, are prefix: the user invokes the commands and then types arguments into a popup prompter. Operation of prefix commands is crippled by particular behavior of popup prompters; once a prompter is visible, it grabs all input. The effect is that strings cannot be copied from elsewhere on the screen (inside or outside BiblioText) into a prompter.[12] This is a failure of *interoperability* [HHN86].

The twin goals of prefix/postfix consistency and interoperability suggest a general approach: choose either prefix or postfix for command operation, and add whatever machinery is necessary for interoperability.

The first possibility is to adopt a uniform prefix model. This requires two changes to the old model. First, commands that operate on refs. must be altered to take arguments (a ref. selection) after command invocation. A simple approach would require typing in the numbers of the refs. to be deleted, but that would sacrifice the directness of pointing. To allow pointing, the command would have to remain suspended until the user explicitly announced that a selection was ready. The second change would require that string arguments be somehow selectable from the screen while the command remains suspended. The SunView selection mechanism would not support this particularly well. To allow argument selection from anywhere on the screen, the SunView selection service must be invoked, but it is fundamentally biased toward the postfix model. For all these reasons, the uniform prefix model is unworkable.

The second possibility, then, is to adopt the postfix model uniformly, like the rest of SunView. This requires only one obvious change, namely that commands taking string arguments now contact the SunView selection service to get the currently selected string, wherever it is on the screen.

A secondary problem arises, though. Often the desired argument string is *not* visible anywhere on the screen, and the user must type it in. Since BiblioText is a read-only browser, it contains no place where the user can type. It is unrealistic to expect the user to type into some other program's window, just so that text can be selected as an argument for BiblioText, so some explicit provisions must be made. The SunView text editor provides a one-line *scratch space* for just this purpose.

The problem with this general postfix approach, even if a scratch space were provided, is that users in our environment do *not* use the SunView editor or anything like it. A new user here, faced with such a program, would have no idea how to begin.

---

[12] A SunView mechanism for doing so has since appeared, but it is a bit obscure and less general than other interwindow operations.

The solution finally adopted for the new interface is an experimental hybrid. Commands will operate postfix; those that take string arguments will use the current textual selection, wherever it is on the screen. If there is no selection, they they will display a prompter as they do now.

This hybrid approach meets the stated goals, and provides a crutch for people who are exploring and don't know what to do. However, the dual prefix/postfix mode of some commands presents potential confusion of its own. Only experience can evaluate the cost of this confusion.

An Accelerator

SUGGESTION: Invoke the command view NOTES by buttoning the display of [notes] in the viewer, rather than by main menu selection. *Decision: Reject.*

This suggestion is completely outside the model currently supported by BiblioText and SunView. The left and middle mouse buttons are only for *selection*, and the right button is only for *command invocation from menus*. The only exceptions are accelerators on panel items and frames, where a left or middle buttons can identify both argument and operation simultaneously, and scrollbars. Adding this behavior to the SunView text subwindow, which now supports the viewer, would require wholesale rework of the operational model and would demand a substantial amount of additional programming.

Clean up Main Menu

SUGGESTION: Rationalize the main menu to make it easier to locate desired operations. *Decision: Use submenus.*

The main menu in the old interface is a jumble of mixed commands. One command has already been removed, Set Working Dir., as described earlier. Possible solutions for the rest include panel buttons, standard keyboard function keys, and submenus.

Many programs, SunView and otherwise, use panel button items for invoking commands (with a left button press). This technique has two costs. The first cost is visual, since too many buttons clutter the panel and create a different version of the same problem (early releases of the SunView DefaultsEditor had this weakness). The second cost is conceptual, since it violates the following (until now implicit) principle: operations that affect the general state of the browser are located in the panel (or frame); operations that affect (or use) the current collection are located in the viewer, directly over the display of the collection.

A second approach is to integrate more closely with the SunView editing model. This requires treating BiblioText as an instance of an editor, a shift in view already under consideration for other reasons (section 5.4 "The Editing Model"). BiblioText can then mimic the SunView editor and use the SunView Standard function keys for those operations that are edits (including, among others L10=DELETE, L8=GET, and L4=UNDO). The disadvantage is that no user in this environment has ever been known to use the SunView editor; the L keys are not labeled on many machines here, and most people do not understand their functions.

This leaves the submenu approach, the one adopted for the new interface. Given the discussion of BiblioText as editor, it is natural to create a submenu labeled edit=>,

where the => is the SunView method for flagging the titles of submenus. Load from File and Store to File likewise migrate into a submenu labeled File=>. Finally, View [notes], View Raw Refs., and View bib Definitions move into a submenu labeled More Info.=>. The standard SunView text subwindow menu is, as before, a submenu labeled textsw=>.

5.3. The Viewer

Two suggestions arose concerning the formatted display in BiblioText's viewer.

Generalize Formatted Elision of Fields

SUGGESTION: Display [W] <locn.> when Verbose Mode, is on, analogous to the handling of [X]. *Decision: Accept and generalize.*

SUGGESTION: Generalize the method for formatting raw refs. When Verbose Mode is off, attempt to display *some* representation of everything present, elided in many cases. When Verbose Mode is on, display as much more as possible, consistent with the goal of convenient browsing. *Decision: Accept.*

The fields %W, %X, and %Z are all formatted differently in the old interface, each a special case. Other defined fields, such as %K, are not formatted at all. In the new interface, this is generalized in several ways.

- First, all other defined fields are included in the formatting in some way; this involved the addition of %K and %S, and might involve more later.
- The elided forms (when the display is not in Verbose Mode) are of the form [%W] instead of [W] (in the old interface) to more accurately suggest their meaning.
- The contents of most fields are displayed when the display is in Verbose Mode, for example [%W]= <locn.>. This applied to all fields except %Z (see below) and %K, since keywords are often long and of little interest. This latter decision regarding %K fields is not necessarily well founded, so it may change.
- The %Z field is still treated specially; it appears as [notes] in either display mode. This field is not part of the standard *bib* format, and its literal contents (file name of external notes) are almost never interesting, since those notes can be quickly displayed with a BiblioText command.

Fine Grained Control of Viewer Modes

SUGGESTION: Allow Verbose Mode to be set independently for each ref. *Decision: Reject.*

The user could be given control over Verbose Mode on a ref. by ref. basis; this would require some command to expand or unexpand the currently selected refs. The drawback would be the increase in the internal state of the viewer, since each ref. could be in one of two modes. Additional formatting clues would be required to identify the mode of each ref., since for some it would not be obvious. More commands would be required to manage this state (for example to set all refs. to Terse Mode, set all refs. to Verbose Mode, in addition to the others). The benefit does not seem to justify the added complexity.

Refwise Selection

SUGGESTION: Alter the feedback response to selection so that highlighting occurs only at the granularity of refs. *Decision: Reject.*

The problem is one of articulatory distance, bridging Norman's "gulf of evaluation". The user selects characters, the viewer highlights the selected characters, but the Current Refs. Selection must be inferred by a simple rule. The obvious solution is to immediately highlight an entire ref. when any character in its display is selected. This solution fails for two reasons.

First, this solution clashes with the improvement to interoperability that now allows SunView selections as arguments for keyword and file operations. These operations take character string arguments that might be *parts* of the formatted display of some ref.; for example, a word contained in a formatted ref. might be the keyword desired for the next index search. Changing the granularity of selection to the refs. would render the other form of argument selection difficult, unless even more complex machinery were added.

One can conceive of solutions in the SunView tradition, but none of them would be easily learned; for example, one could select refs. by holding down the shift key while buttoning. Furthermore, these solutions would be difficult to implement on top of the standard SunView text subwindow.

This remains an unsolved problem, a clear weakness in the interface.

5.4. The Editing Model

The most far reaching suggestions concern the basic operating model of BiblioText.

Cut/paste editing

SUGGESTION: Complete basic editing functionality to allow arbitrary rearrangement of the collection. *Decision: Postpone.*

As the analysis made clear, BiblioText already supports a subset of basic editing functionality on the current collection. It might be helpful to complete that functionality, and it might become more understandable in the process.

Since a list is a simple editing domain, it seems reasonable to design a simple editor for lists. Many of the simplest editors use *cut-paste* model; this one would work similarly, but at a coarser granularity than text editors. It would require some addition to BiblioText's functionality: replace delete with cut (to clipboard), add paste (from clipboard), and add, just for good measure, an additional More Info. command called View Clipboard.

I designed these changes and sketched out the semantics of these new commands. Only small issues arose during this design:

- a paste should empty the clipboard;
- paste should insert *ahead* of the first ref. in the current selection, and should *append* if there is no selection; and
- the contents of the clipboard should *persist* across all editing operations, including Delete All.

The similarity of the SunView global selection model (there is something called the *shelf* that is like a clipboard) suggests closer integration. For several reasons, however, this is not practical: the semantics of SunView editing might clash because of the change in granularity; no local users understand the full SunView global selection model; and I don't understand the full SunView global selection model.

Time constraints prevent the implementation of this suggestion.

Undo Editing Changes

SUGGESTION: Make all operations on the collection reversible. *Decision: Postpone.*

This is an excellent idea, one that becomes more important when the general editing model is accepted.

I designed an Undo command based on the following simplifying assumptions:

- Undo only those operations that change the *contents* of the collection; viewing changes (scrolling, changing modes) are transparent;
- Undo can only unwind *one* previous operation; and
- Undo, since it changes the contents of the collections, is itself *undoable*

Thus, two Undo operations in succession will cancel one another (behavior familiar to users of *vi* [Joy79]). Although the internal implementation has some interesting twists (a second, invisible clipboard, and a couple of new, invisible, operations must be added), it is relatively straightforward.

More interesting is the presentation of this function to the user. This implementation could be explained to the user at the same level of abstraction as the existing editing operations, using concepts like clipboard, cut, and paste. This would be like Young's register model of the stack machine [You81]. But I imagine most users would be much more satisfied with a model more like Young's model of the algebraic calculator, namely a simple statement: "reverse the action of the last change to the collection."

One final interesting piece of semantics arose during this deliberation. In the conceptual model the Sort command appeared a bit ambiguous; it could either be viewed as an edit or a viewing mode, depending on how one defined the collection (section 3.5 "Thoughts on Model Building"). Once an Undo operation is available, however, the choice becomes clear. I designed semantics for Sort and Undo based on both assumptions. If Sort is considered an editing operation, the semantics of Undo are natural, just what a user would expect. If, on the other hand, Sort is considered a viewing operation, then it must be transparent to Undo; one should be able to undo an operation performed before an intervening Sort. In this case, however, the necessary semantics are unintelligible. So, even though SortOrder still appears on the panel, with the other modes, it logically must be an edit operation.

Time constraints prevent the implementation of the Undo mechanism.

5.5. Other Design Issues

Two final suggestions fell into none of the above categories.

Output Formats

SUGGESTION: Create and support some specialized storage format(s) that can be read, maintained, and stored directly by BiblioText; make the storage external to the *bib* database so no indexes are affected, and refer to documents in the database with imprecise citations. *Decision: Reject.*

This seems attractive, but I have been unable to imagine a format that would be of general use and would not be bound to one user's personal way of working.

Multiple Auxiliary Frames

SUGGESTION: Allow multiple files of external notes to be viewed/edited simultaneously. *Decision: Reject.*

Unfortunately, this suggestion cannot be supported. SunView software restricts the number of screen objects in a single process, and multi-process programs in the UNIX environment present formidable technical problems that would cripple any such design.

5.6. Thoughts on Design Complexity

This phase of the experiment began with a set of suggestions for change to the old interface. Every suggestion appeared clear and promising in its original context, an evaluation from some point of view. In the context of the whole design, however, surprisingly few were clear, and almost none could be adopted in isolation from the the other suggestions and from the original, high level goals of the program.

Complex interdependency among the issues arose locally, among various design decisions for the program itself, and contextually, between design decisions and external aspects of the program's construction and use. Once again, the problem of context confounds everything; nearly every suggestion required some consideration of the contextual influences listed earlier (section 4.6 "Thoughts on Theoretical Evaluations"). Most of the decisions result from compromise of some kind; in some cases no workable compromise seemed possible at all.

6. An Empirical Evaluation

This section begins the third phase of the experiment, an evaluation of BiblioText's redesigned user interface and of the process that led to it. This evaluation is based on version 3.4, referred to as the *new interface*; figures 5.1 and 5.2 in the previous section show the new interface in operation.

This phase begins with an informal, empirical evaluation of the new interface based on sessions with new users. Two colleagues, whom I'll call Samuel and Alan, each agreed to spend an hour or two learning to use BiblioText and making comments. Both subjects had extensive experience with UNIX, the Sun workstations, and *bib*, but no previous exposure to BiblioText.

The first two parts of this section introduce the study and its participants. The next five parts discuss the results of the sessions, each part treating a group of related issues: personal working contexts, interaction with the window system, BiblioText's viewer, basic functionality, and general reactions. A final part reports general observations from

the evaluation.

6.1. The Study

The setup for the study included:

- a generally accessible, installed version of the program;
- a *man* page for BiblioText, five pages of online summary documentation in the style to which UNIX users are accustomed;
- typeset copies of the documentation;
- two chairs at a Sun-3/75 workstation in an office where there would be few disturbances; and
- a tape recorder.

The subjects made no particular preparations for the sessions.

After each subject had arrived and logged onto the workstation, I helped set up an appropriate working environment. This environment consisted of two contextual layers: internal BiblioText configuration and a general working arrangement on the screen.

Internal BiblioText configuration provides access to the subject's own collection of *bib* data files, indexes, and personal macro (abbreviation) definitions. Configuration data reside in the SunView *defaults database*, accessible via the interactive program *DefaultsEditor*. Unfortunately, BiblioText cannot read this data interactively, so some encounter with *DefaultsEditor* is prerequisite to any first use. I assisted both subjects with this prelude, since both were unfamiliar with BiblioText and only slightly familiar with the *DefaultsEditor*. Neither subject cared to read the documentation in advance, although this would have helped them with this step.

I then made suggestions for a working context: start BiblioText, start an editor of some sort, and prepare to edit a typical document that involves *bib* citations. Samuel used *emacs* and Alan used *vi*.

During the remainder of the session, I allowed the subjects to experiment with BiblioText as they chose. I answered questions, offered explanations when they were obviously confused, and prompted comments as they worked. Later in each session, I steered each subject toward parts of BiblioText's functionality that they hadn't yet explored.

6.2. The Subjects

Other than willingness, Samuel and Alan were chosen as subjects because they are heavy users of both the Sun workstations and *bib* software. Both have personal, growing collections of indexed references; Samuel is currently writing a dissertation and Alan is writing a Qualifying Paper.

Samuel

Samuel has been collecting and using *bib* data for a long time; he began years before there were any workstations. He has overlaid incredibly complex functionality onto *bib*. He uses it as a database at several levels; most interestingly he has built a facility for automatically generating forward and backward references to sections, figures,

and tables in a *troff* document (he has programs that build a small *bib* database for each chapter).

He organizes his bibliographic data and related notes much differently than the standard, with multiple indexes and *makefiles* [Fel79] that automatically pass data through four or more stages of processing. In particular, he has built extra machinery to guard against imprecise citations losing their uniqueness as his collection grows.

Samuel ceased using SunView software six months earlier; he now uses the X window system [Get86] on the Sun workstations. He commented at the beginning of the hour that he has forgotten most of the details about using SunView. This turned out to be true at both the conscious and motor learning levels.

Alan

Alan has been a heavy user of the workstation for some time; he still uses SunView at least part of the time. He doesn't consider himself a student of its advanced features, though.

He has recently started using and collecting *bib* data. He is preparing a Qualifying Paper and is beginning to use the *bib* programs for managing his own library and research notes.

6.3. Personal Working Contexts

This part of the section begins the report on the sessions, based on written notes and tape recordings. The set of issues here appeared almost immediately, before the subjects had much exposure at all to BiblioText. These issues concern variations in personal working contexts between the two of them (and between them and myself). These turn out to have significant effects on their experiences with BiblioText.

Personal Databases

Variations became clear immediately as I helped each subject set up configuration information beforehand.

Samuel has *bib* refs. scattered among several locations, maintained by a complex of makefiles, *sed* and *awk* scripts, and custom batch programs. He creates each new raw ref. in a separate file along with associated notes, and has the collection of refs. built automatically for indexing. He maintains several indexes, based on different definitions of keywords; only one or two of them appeared suitable for browsing with BiblioText. Because of this arrangement, he has no internal notes (no %X fields).

Samuel uses %X fields for location information. He groups refs. into files according to subject and to whether he owns a copy or not.

Alan maintains only one file of raw refs., but he already finds it unwieldy. He keeps two copies of the raw file, one for editing and one that is currently indexed; this allows him to modify the file while using *lookup* concurrently.

When Alan copies his working version of the refs. to the final version for indexing, he passes the file through a batch program that inserts extra location information based on table lookup about journals in his library; this gives him current information about whether any given article is in his journal library. He uses the %Z field for locations

(which he selected randomly, since it is not defined in the *bib* standard); of course, this clashes with usage in BiblioText. Furthermore, he inserts multiple copies of this field; BiblioText was designed with the assumption that there would never be more than one, so only the last one in each raw ref. gets formatted.

Editors

Samuel uses *emacs* and Alan uses *vi*. The variation can cause problems when doing interwindow copies of text. For example, *vi* has separate command and insertion modes; in the former mode it reacts badly to a sudden stream of characters from the selection service.

Learning & Exploration

Neither wanted to read the printed documentation, either in advance or during the sessions. Both wanted to experiment and allow me to offer explanation on the fly. Both were happy to explore the menus to see what was available.

6.4. SunView Standards and Tools

Both subjects frequently became confused and needed help. Most instances concerned low level interactions with the window system and interface artifacts in BiblioText. Both commented frequently on the difficulty they have using the SunView system, and repeatedly criticized it on small points. This section summarizes their points of confusion and their complaints.

Frame Operations and Accelerators

Neither subject was familiar with keyboard accelerators for standard window operations; for example control-middle-button starts a window resize, an operation that otherwise requires a menu invocation.

Samuel in particular, not having used SunView for some time, had to be reminded frequently which button to use for menus, how to close and open windows, and the like. There was also some initial confusion about the assignment of keyboard focus. All are handled differently in the X window system. X is based on a somewhat different underlying model, so the problem is deeper than simple reassignment of commands.

At one point, Samuel tried to get the viewer's scrollbar to work, but he couldn't remember how to use the small buttons appearing at either end of the scrollbar. I couldn't help, because I can't remember how to use them either.

SunView Selection Service

Neither subject understood the SunView global selection service. Neither was aware that this service involves several levels of selection (BiblioText only uses the primary selection). Neither knew how to use keyboard accelerators for copying. Alan knew how to use a small subset that involves the primary selection and the Stuff operation. Stuff is a special copying operation that works only among TTY emulator windows; it is present in the 3.2 release of SunView for backward compatibility with earlier releases, but it is not part of the current global selection service.

This lack of expertise was not a problem for argument selection to operations like keyword lookup, but it did present problems when the subjects tried to copy newly created citations into their document editor windows. This problem will be described in a bit more detail below.

Samuel had trouble with selection, even after I explained it, since the X model is somewhat different.

Unintended Selection

Both subjects had trouble with unintended primary selections that persisted. This interacted badly with the new hybrid model for commands that take string arguments. At one point Alan wanted to perform a keyword lookup with new keywords. He initiated the command, but no prompter appeared; instead, the command ran and failed to find anything. Alan had no idea what had happened. There was an active primary selection elsewhere on the screen, and the lookup operation had used that string as its argument string; none of BiblioText's visual feedback, however, helped identify what had happened.

There seems to be a three-way clash in models for argument specification: postfix command with selection (SunView model), popup prompters (prefix model), and the retention of the last string used for a default (still in place from the original design, the default appears in the prompter). This is greatly exacerbated by the primary global selection, which can be created unobtrusively during interactions with other window, even interactions that involve no selections.

Prompter Input

Samuel complained repeatedly about local editing conventions in popup prompters. Control-U erases the whole line in the prompter, but he is familiar with a different convention in X. He types quickly and made this mistake repeatedly.

Panel Item Menus & Accelerators

Both used mouse buttons to explore panel item behavior. Both began with the left button. Only the Sort item responds to a left button, but when the collection is empty it does nothing; thus, they found the panel apparently unresponsive.

Later however, both at various times accidentally hit Sort with the left button and invoked "accelerator" behavior, namely to advance the current selection to the next one in the list. The result was a visible sort operation (according to the message field) but no feedback from the panel item itself. Both were confused by this.

Alan persistently forgot to use the right button on panel items. I finally asked about this, since right button menus are the SunView standard. He commented that the right button seems natural for menus outside the windows or in the text viewer, but he just couldn't remember it in the panel. After some discussion, we realized that he uses the *gremlin* drawing program much more than any other SunView program; contrary to SunView standards (it predates them) the *gremlin* control panel requires left button to initiate actions, right button to print additional help information.

Samuel criticized the toggling behavior of panel menus every time he needed to use them. In particular, they only allow one action (the toggling of one selection) per menu invocation. This makes it time consuming to effect multiple changes to the set of active indexes. Samuel experimented with likely accelerators that didn't work; for example he tried a left button selection while keeping the menu displayed by holding down the right button.

Panel Text Item & Selections

Both subjects, especially Samuel, became confused when trying to select newly created citations in the panel. The item is displayed in a "panel text item" which supports selection with curious, partial semantics. For example, the item can sometimes be highlighted and not be the primary selection, and, unlike other text in SunView, it gets selected in its entirety with only one left button press. Samuel tried to type into the item at one point (without luck), since there seemed to be no reason not to; at the time he was trying to type in keywords to select as arguments to a lookup command.

Samuel finally commented that there should be a text scratch pad for use with this postfix command arrangement.

Textsw Functions

Both subjects had occasion to explore the functionality of the viewer text subwindow. Samuel found Split on the textsw=> submenu, and split the viewer into two parts for an imaginary (but plausible) task he wanted to perform. He then added some new refs. to the collection and found that both views had been repositioned to the first new ref., not what he wanted or needed. I hadn't taken this possibility into account, because I knew that the SunView split operation had potentially fatal flaws in the current release.

Alan also explored the textsw=> submenu and asked several questions about it. He was completely unfamiliar with it.

Both had occasions for textual search through the viewer, but didn't know how to do it. I suggested the use of Find in the textsw=> submenu, but neither was able guess how to make it work. I walked both of them through it several times. Samuel wondered why he couldn't just type control-S over the viewer, "like in EMACS."

Even the scrollbars were troublesome. Alan complained that the bubble does not get repainted until the cursor moves into it; this can be misleading when the viewer is repositioned under program control. None of the three of us know how to use the buttons appearing at the ends of the scroll bar. I have studied the model several times, but I find it difficult to remember.

Auxiliary Frames

Alan had previewed a text file with citations, and was reviewing his error messages in an auxiliary frame. He needed to look at some raw refs. to fix one of them, but lost his error messages when the raw refs. display took over the auxiliary frame. Because of unfortunate SunView implementation restrictions (file descriptors), BiblioText creates only one auxiliary frame that is shared among various requests for additional information.

Alan wondered how to retrieve his list of errors; I suggested that he repeat the file loading operation to recompute the error messages.

6.5. The Viewer

Formatting

The exact details of the formatted display were the subject of some discussion. Both were comfortable with the elided fields, but Samuel had comments about the techniques for their expansion; he didn't find the present arrangement obvious.

Both were interested in seeing %K fields expanded when in Verbose Mode but agreed that it made no sense to expand [notes], since the file name would offer no new information.

Samuel wanted to expand individual refs. into verbose mode. He also asked about expanding individual fields, perhaps into a separate window. He wanted to point and button objects that would respond by expanding.

Selection

Samuel encountered a subtle problem with selection. He tried to select an entire ref. (not realizing that a single character in it would suffice) and managed to include the single character on the blank line that separated the ref. from its predecessor in the collection. BiblioText interpreted this character selection as overlapping into the previous ref.; when Samuel requested a citation, he was presented with one for the ref. just before the one he thought he had selected.

Samuel became accustomed to the global primary selection for string arguments, but then was confused by commands that operate on refs. He selected text elsewhere on the screen and tried to request a citation. The error message no current selection appeared in BiblioText, but he was confused since there clearly was a selection; there was a global SunView selection but no ref. selection in BiblioText.

Positioning

Both were at first unaware that BiblioText appends newly acquired refs. to the current collection; this confusion persisted, apparently because BiblioText in each case repositions the viewer to the first new ref. added. Alan asked himself out loud about this, examined BiblioText for clues (the number of refs vs. the number that had been added, the sequence number on the first ref. showing) and concluded that they are appended. Samuel appeared to be uncertain about this for some time, until we discussed it.

At one point, Samuel performed a lookup that matched two refs.; neither was new to his current collection, so they weren't added. He was surprised that the viewer wasn't repositioned to one of the old ones.

6.6. Functionality

The issues in this section concern basic functionality of BiblioText and are less affected by surface issues.

Read-Only

Both subjects were at first unaware that the browser can only read *bib* data, not modify it. Samuel temporarily confused citation with reference and tried to create a new ref. After a short explanation, both were comfortable with the restriction since they know *bib* and its limitations.

Additions

As mentioned earlier, both subjects were uncertain about the policy of adding newly acquired refs. to the current collection. Samuel expected that they would replace the current collection.

Keyword Lookup

Samuel selected a *multiple imprecise citation* from his document and initiated a keyword lookup. This is perfectly legal *bib* usage and is in the form

[.<keywords>,<keywords>.]

It results in two citations in the finished document. Unfortunately, the BiblioText keyword lookup function does not respect this convention; it treated the entire string as one keyword list and failed to find any matches in the indexes. There was no feedback to suggest why the lookup failed.

Make Citation

Both subjects were sensitive to problems of uniqueness in citations; both wanted to know "how unique" the newly created citation is. I explained the heuristic BiblioText uses, and both agreed that it is reasonable for an automatic tool. Samuel observed that he adds extra keywords just to make sure that later additions his ref. collection don't create ambiguity in existing citations.

SORT Function

Neither found much use for the Sort command. I suggested using it several times in situations where they needed to locate something, but it isn't always enough. For example, the papers by specific author don't always appear together, unless the author is always first author.

Alan tried to sort an empty collection and wondered why it didn't appear to work; the current sort order is defined to be Unsorted when the collection is empty, so it can't be changed.

Edit Functions

Neither subject could imagine wanting to do much editing, since it wasn't clear what long term value a collection could have. Consequently, neither felt the need for Undo, although Samuel asked about it at one point.

File Loading

I steered both subjects toward loading two different kinds of files (raw refs. and text with citations). Samuel later tried loading an index file, reasoning that the two possibilities he had already seen might generalize to many; he wasn't disappointed that it didn't work.

Both realized immediately what BiblioText does when loading a text file with citations, apparently because of the step by step visual feedback of the form

checking line <n> [.<keywords>.]

for each citation processed.

Alan found this document preview extremely helpful, and discovered citation errors in his current paper. He found the error messages self-explanatory. He was able to improvise with BiblioText to correct the problems and insert correct citations.

Samuel commented that "it did the right thing."

Both were a bit uncertain about the detailed semantics of the document preview. I explained that it attempted to mimic *bib*, with all its peculiar semantics about what is a multiple match (an error) and what is not. As experienced *bib* users, they found this explanation satisfactory.

Set Working Directory

Samuel encountered faulty behavior trying to reset his working directory; this was apparently caused by a primary selection on the screen that he did not see. The problem seemed to be a subtle interaction with the UNIX system interface that I don't understand. He was able to try again and correct the problem.

New Tasks

Both subjects suggested new tasks for which they would use BiblioText if it had the right functionality.

For example, Alan asked "I've found a ref. in your index, how do I ask if it's also in mine?" I had to help with this one, since it isn't natural: turn off all indexes except his own, select a string of text from the ref. that has enough words in it to identify it (punctuation and short words get discarded), and perform another keyword lookup.

Samuel also wanted to perform tasks associated with index maintenance. He wanted to ask if a ref. is in more than one index. The BiblioText policy of eliminating duplicates in the current collection works at cross purposes here. Newly acquired duplicates leave no record.

Both of these new tasks were a bit beyond the basic model on which BiblioText was built. Both subjects agreed that the basic BiblioText are desirable for some tasks, but not for others that they invented.

Main Menu

Alan found the File=> submenu (containing Store to File, and Load from File) a bit too cryptic. He asked "what are we loading and saving?"

Samuel and, to a lesser extent, Alan were confused by term cut. Samuel didn't seem familiar with cut/paste terminology at all. A cut operation resulted in a message using the word deletion, confounding things even further. I finally explained in *emacs* terms, but the concept of the clipboard remained murky.

Prompter

Both wanted to copy text into a prompter from elsewhere, and complained about prompter behavior (it freezes the display and preempts all input). Samuel wants to start an operation, move to another context on the screen, and then return.

Samuel complained about the local editing conventions in prompters, since it differs significantly from those used in X.

View bib Definitions

Neither had trouble understanding the result of a View bib definitions command. Both commented that it appeared extremely handy (even though it is the operation conceptually farthest away from the basic model).

6.7. General Reaction

Both subjects, despite confusion and repeated criticism of low level interaction techniques, were impressed by BiblioText. Samuel commented that if he were just starting with his library and database, he would use this tool and organize his data differently. He has too much time invested now, he's in the final six months of writing his dissertation, and he doesn't feel justified in retooling. As things stand now, his data model differs too much from the standard for BiblioText to help him.

Alan, on the other hand, is just starting. Already during our session he began to plan a restructuring of his data that would take advantage of the browser. He mentioned that BiblioText directly addresses at least one problem with which he had been grappling. He commented the next day that BiblioText had already repaid the time he spent during our session.

6.8. Thoughts on Usability

All three of us (the two subjects and myself) have superimposed our own functionality onto the suite of *bib* tools, using it to implement different kinds of databases. To the extent that our models were similar (or malleable), BiblioText is useful. Otherwise, it appears to be of little help.

There are incredibly many relevant details outside the conceptual model of BiblioText; most complaints concerned low level interactions, and to a lesser extent the personal use of bib formats.

One criterion for a "feeling of engagement" is that "The interface should be unobtrusive, not interfering or intruding..." [HHN86]. The empirical evaluation suggests that

BiblioText does not succeed, at least not for the two subjects involved. Nearly half of their comments concerned minor points of interaction, hardly an unobtrusive interface.

7. The New Interface Reconsidered

This section continues the third phase of the experiment, an evaluation of BiblioText's redesigned user interface and of the process that led to it. This evaluation is based on version 3.4, referred to as the *new interface*; figures 5.1 and 5.2 in an earlier section show the new interface in operation.

The third phase began with an informal, an empirical evaluation of the new interface, based on sessions with two new users (section 6 "An Empirical Evaluation"). This section section revisits each group of design decisions made during the second phase of the experiment (section 5 "Redesign") and judges the success of each in light of the empirical evaluation.

Some design decisions from the second phase appear, in these judgements, to be faulty or incomplete. When the discussions below lead to ideas for correcting them, these appear as suggestions in the same format as those produced during the first phase of the experiment. Some of these final suggestions eventually found their way into BiblioText, after the completion of the experiment. Figure 7.1 below shows a much later version, the one described by BiblioText's user manual [Van88].

7.1. Cosmetics and Visual Consistency

The decisions in this group were generally successful. Some problems surfaced where the user is not being given enough information. Other problems, artifacts of SunView support, had been anticipated but didn't admit to easy solution.

Clearly Successful

The problems addressed by these three decisions never arose during the sessions, and they provoked no comment from the two subjects.

- *Active vs. Nonactive Panel Items.* Both subjects were drawn to the first line of the panel when attempting to find mouse-sensitive items. Neither tried to initiate activity from the two inactive items (other than one attempt to type into the panel text item that displays messages).
- *Auxiliary Frame Labels.* Neither subject had difficulty identifying auxiliary frames with BiblioText, and there was no confusion about the contents of an auxiliary frame.
- *Secondary Frame "Done" Buttons and Menus.* Both subjects were able to dismiss auxiliary frames without hesitation.

More Information Needed

The next decision gives the user more feedback than before, but experience suggests that yet more is needed.

- *Signal When Working.* First, both subjects became confused by sort operations when they accidentally triggered the panel item's left-button "accelerator" behavior; the

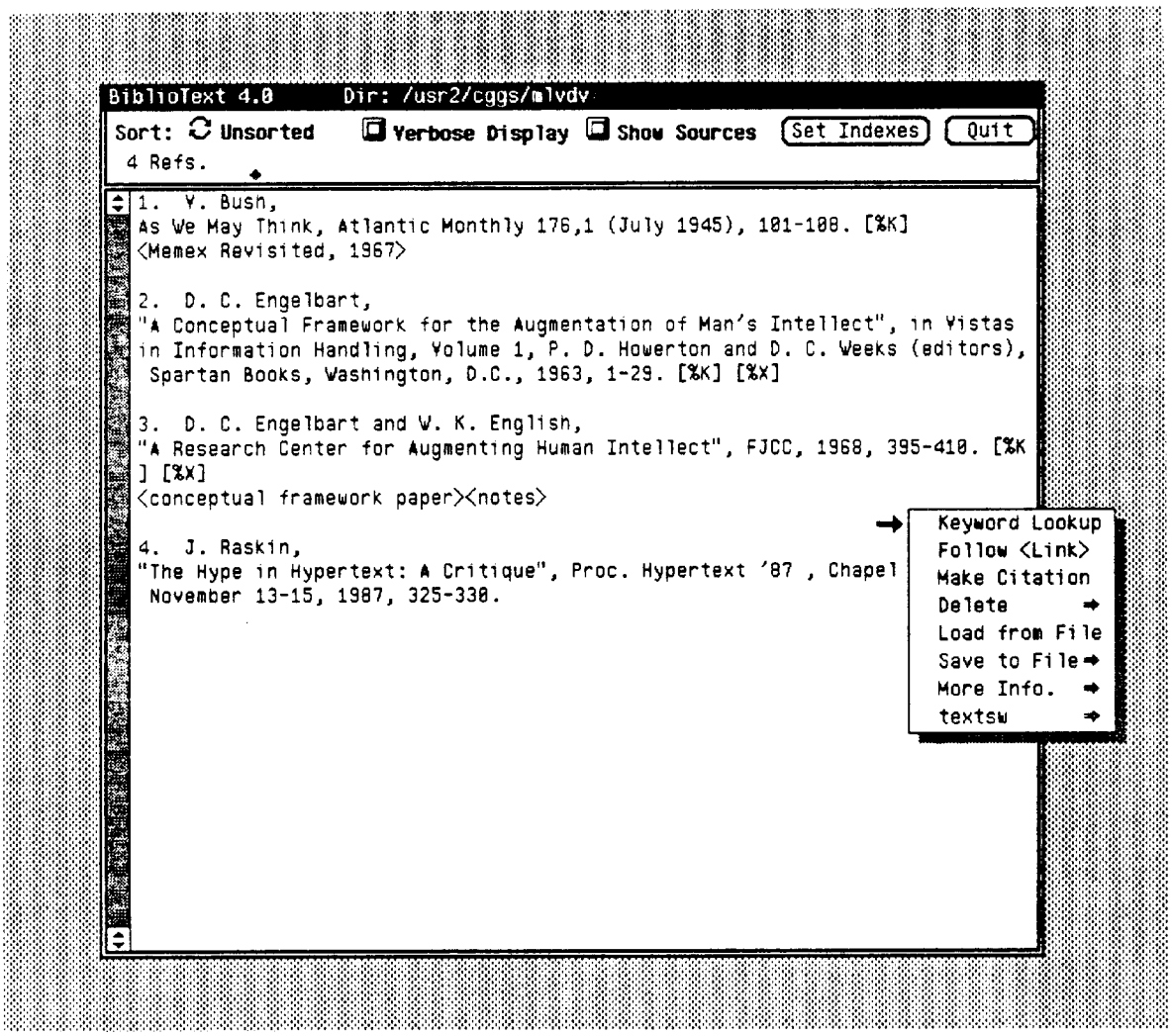

Figure 7.1
A much later version: main menu visible

message

sorting, please wait...

without them having made a menu selection. Second, several keyword lookup operations failed when there was an accidental primary selection elsewhere on the screen. It appeared to the user that the search had started without requesting keywords, and the resulting messages,

no keywords supplied

or

0 Refs. found,

gave no further clues.

More feedback would help, enough to help the user understand in each case exactly what operation BiblioText is attempting. This is especially true when a global selection may be taken as a string argument.

SUGGESTION: Give the user more specific feedback for operations that take string arguments: for sorting, which sort order is the target

sorting by Author-Date, please wait

and for index searches, which keywords were used

14 Refs. found, 10 new for keywords [.<keywords>.]

SunView Clashes

For two of the design decisions, anticipated clashes with the SunView model did turn out to be troublesome. No easy solutions are apparent.

- *Active Panel Item Consistency*. Both subjects tripped over the "accelerator" behavior of the panel Sort item, accidentally hitting it with a left button press and then not knowing what had happened.

More feedback will help, as would more familiarity with SunView, but the problem remains for new users in this environment.

- *Current Working Directory*. As predicted, neither subject thought to investigate the frame menu when they wanted to change the cwd.

This only affects new users, and no obvious solution presents itself that doesn't have other costs.

7.2. Command Formation

The decisions in this group were clearly helpful, but the issues are far from solved.

- *Interoperability and Command Syntax Consistency*. Both subjects had immediate use for making screen selections as string arguments to BiblioText commands, and found it natural to do so; this change seems to be helpful. On the other hand, there were several instances of confusion about string arguments.

The possibility of this confusion, which appeared as "a three-way clash in models for argument specification" (section 6.4 "SunView Standards and Tools"), was anticipated at the time the experimental hybrid model was designed: "... the dual prefix/postfix mode of some commands presents potential confusion of its own." (section 5.2 "Command Formation"). Since this does appear to be a problem with users, more work needs to be done. The earlier design discussion suggested alternatives, but no choice is obviously best. For example, one alternative presented was:

> SUGGESTION: Eliminate popup prompters, add a one line scratch area, and embrace completely the SunView postfix argument model.

The drawback is that users in this environment are unfamiliar with the model. It might also be awkward to repeat commands using the same arguments, but it is not clear how important this for most patterns of use.

- *Clean up Main Menu.* Both subjects were comfortable with the menu organization in general; neither expressed confusion about the use of submenus. However, there was some confusion about what functionality corresponded to the menu items.

The problem wasn't that the menus are unintelligible; they simply don't display enough information to educate new users in some aspects of BiblioText's functionality (especially file operations). This is an instance of a familiar tradeoff; expanded menu titles to help new users would be clutter to experienced users.

7.3. The Viewer

This group of decisions provoked more comment from the subjects than any other, but fewer errors. This may be because the visual display is the most immediately perceived aspect of the interface. Some comments touched on suggestions that had been rejected in the redesign, so those suggestions have been resurrected here for further consideration.

More Generality Needed

The following decision was successful, but didn't go far enough.

- *Generalize Formatted Elision of Fields.* Both subjects use various fields in different ways than I do and had occasion to examine the contents of fields that had previously not been displayed in the viewer. The display of fields, both elided (appearing as [%W]) and expanded (appearing as [%W]=<locn.>) was self evident to both subjects; both made appropriate inferences about the display, without comment. The arbitrary design decision not to expand %K fields in Verbose Mode was wrong; both subjects found the irregularity confusing and at times wanted to see the contents of %K fields. Both agreed that [notes] is an appropriate presentation of %Z fields under the circumstances. One subject uses multiple %W fields in his data; BiblioText, on the assumption that there is only one per ref., displayed only the last.

The decision to generalize was correct, but I still imposed too many unwarranted assumptions that were based on my way of organizing *bib* data. The variations that appear in this sample of three users suggest even more generality.

> SUGGESTION: Include all possible fields in formatting, including those not defined in the *bib* standard (lowercase too). Support multiple instances of all

fields. Adopt a uniform formatting convention for all ''miscellaneous'' fields; for example, when Verbose Mode is off, display only a single field label as before (appearing as [%W]), and when Verbose Mode is on, display each field on a separate line, with instances separated by commas (appearing as [%W]=<locn.>,<locn.>).

Originally Rejected

Both of the two remaining decisions in this group were rejected during the redesign, but were the subject of some comments by the subjects.

- *Fine Grained Control of Viewer Modes.* One subject wanted to set Verbose Mode for an individual ref. in the collection, and later wanted to expand the display of a single field.

The original suggestion, to allow Verbose Mode to be set independently for each ref., was rejected as too costly in complexity, both in implementation and interface. This experience suggests that the added complexity in the interface might not be so bad, since one subject, at least, expected the function to be available. The SunView implementation environment, however, doesn't adequately support the fluid style of interaction that these suggestions would demand.

- *Refwise Selection.* Both subjects were uncertain about the relationship between character selection in the display and the ref. selection necessary for BiblioText commands. One was surprised when a selected character on a blank line was interpreted to include the preceding ref.

This kind of difficulty was predicted repeatedly, both by experience and from various analyses. The suggestion to make viewer selections operate at the granularity of refs. was rejected for reasons that are still valid, however. It remains, as noted during the redesign, "a clear weakness in the interface." (section 5.3 ''The Viewer'').

New Functionality

Finally, one issue arose during the empirical evaluation that fell outside suggestions considered during the redesign, but which had manifested itself earlier during construction of the conceptual model.

Both subjects were interested in knowing about refs. added from multiple sources, typically when discussing ''new tasks'' such as index maintenance. BiblioText only records the Ref. Source and Cite Source file names when a ref. is first added to the collection. If duplicates appear during later acquisitions, any new information is discarded. This behavior was mentioned earlier among other ''Minor bugs in model'' (section 3.5 ''Thoughts on Model Building''), ''... this doesn't always have a clear analogue in the conceptual model.'' It also turns out that this design isn't the most useful one.

SUGGESTION: Extend the collection so that each ref. can have any number of associated Ref. Source and Cite Source file names. Display all in the viewer when Sources Mode is on. Whenever a ref. is acquired that it is already present in the collection, add its Ref. Source and Cite Source file names to the existing ref., if not already there.

7.4. The Editing Model

Two significant generalizations of the BiblioText editing model were designed but not implemented because of time constraints: cut/paste editing, and undo. To suggest this projected functionality, new commands were added to the reorganized main menu, plainly marked as unimplemented.

- *Cut/paste editing*. Neither subject saw much use for generalized editing. One subject was confused by cut/paste terminology, apparently independently of its implementation status.
- *Undo Editing Changes*. Both subjects were familiar with the concept of an Undo command, and one subject spontaneously asked if it is available in BiblioText. Neither seemed to need it, because they didn't attempt much editing.

It seems that expanded editing functionality by itself buys little, and that cut/paste terminology didn't contribute. Editing would only be important to users if they perceived some longer term advantage to maintaining a collection. Thus, this aspect of the redesign interacts strongly another issue (''Output Formats''), discussed next.

7.5. Other Design Issues

A persistent issue, appearing in several threads of analysis and design discussions, suggests enhancing functionality so that collections can be annotated and made to persist in useful ways.

- *Output Formats*. Both subjects asked in what form the collection is written to files; neither could see much use to keeping them in their current form.

The suggestion to extend output formats was rejected during redesign for lack of good ideas. Any change will have to be made as part of a general extension to the concept of what BiblioText can do, such as the ability to directly help with the creation and maintenance of annotated bibliographies. Only then will general editing functionality become important.

Task analysis led to the following suggestion, but it had to be rejected because of inadequate SunView support.

- *Multiple Auxiliary Frames*. One subject, while correcting citation errors that he had discovered with a preview, requested some raw ref. data and lost his error list.

This experience suggests that the suggestion is valid. Unfortunately, there is no solution at this time.

8. Conclusions

This experiment led to insights and conclusions at varying degrees of generality. This section reviews some of them, beginning with the most specific result, an improved version of BiblioText. The second part reviews lessons that became apparent during individual phases of the experiment, and the third draws together general themes that emerged throughout.

8.1. Improvements

The most concrete result of this experiment is version 3.4 of BiblioText. The new interface developed during this experiment is clearly superior to version 3.3, the old interface. These improvements resulted from various principled approaches and otherwise might not have appeared on BiblioText's evolutionary trajectory. Furthermore, additional improvements were left pending at completion of the experiment: some inspired by the evaluation of the old interface (section 4 "Theoretical Evaluations") but deferred for lack of time, and others prompted by the evaluation of the new interface (section 7 "The New Interface Reconsidered"). Some of these additional suggestions eventually found their way into BiblioText, version 4.0, current at the time of this report [Van88].

Almost as important is the small user community that appeared as a side effect of the empirical evaluation. BiblioText was installed locally, acquired a UNIX style *man* page (derived from the conceptual model developed during the experiment), and began to attract further constructive comment.

8.2. Lessons

This experiment explored several approaches to the design of user interfaces. The summary here recapitulates the three experimental phases, describing some of the insights, or lessons learned at each step. It would be more accurate to describe these as lessons *re*learned, since most are not particularly new in the abstract. However, their appearance in the context of work on a real system of moderate complexity demonstrates the difficulty of bringing theory into practical use.

Phase I: Evaluate the Old Interface

In the first phase of the experiment, I evaluated BiblioText as it existed at the beginning of the project (version 3.3). The evaluation proceeded in two parts: a set of low-level descriptions (section 3 "Descriptive Models") and a series of more abstract analyses of the interface (section 4 "Theoretical Evaluations"). The latter analyses attempt to follow advice implicit in selected writings in the research literature on human computer interaction. Theoretical points of view considered in this phase include task-based design, conceptual models, metaphor, and direct manipulation.

Lesson 1. *Complete functional specification of a real mouse and window based program is impractical, perhaps impossible.*

I failed to complete an attempted functional specification. One problem was the profusion of detail, resulting from the many contexts in which BiblioText is situated: mouse and screen object behavior (from the SunView toolkit); interaction with other window programs (via the SunView selection service); the semantics of indexing and citation (from *bib* data and programs); operating system behavior such as file naming conventions (from UNIX), and others. Attempts to simplify by separating functionality and interface also failed; the distinction is blurry and not well-defined for this kind of program.

Had I completed the specification, its volume would render it unintelligible. And even if it could be made intelligible, it would fail to capture the attributes most obvious

to a user: *style*, *attractiveness*, and *feel* for example. In the end, I retreated to a few partial descriptions that contained just enough detail for the tasks at hand.

Lesson 2. *Conceptual models can be helpful, both for the designer and for explanation, but they do not necessarily exist a priori for working, useful programs.*

It was surprisingly difficult to articulate a coherent conceptual (*surrogate* or *structural*) model for BiblioText, especially since I had assumed the implicit existence of such a model. After all, I understood the program, and coherency and simplicity had been design goals all along. But the model demanded that I invent concepts outside my prior view of the program, and the final result still didn't match the program completely.

On the other hand, constructing this model predicted (correctly) some areas of potential confusion. The model also helped organize documentation for users, and it later served as helpful guide when I evaluated various suggested changes to the interface.

Lesson 3. *The research literature on user interface design is only marginally relevant to practical, mouse and window based programs.*

In every case, contextual realities confounded the utility of principles. The conceptual model mentioned above was helpful, but necessary restrictions on the program made exact fit between program and a clean model impossible. Most discussions of metaphor concentrate on naive metaphors, how computational objects are like non-computational objects. But users in this environment are not naive, and they are more likely to consider a new system in light of other computational systems than to non-computational objects. They also bring concepts, skills, and prejudices, developed through experience with other computational objects. Direct manipulation interfaces promise some advantages, but in at least one area, BiblioText is useful precisely because it does not work by direct manipulation. Finally, it is difficult to know what *consistency* means when applying these techniques; in may cases the answer depended on assumptions about the context in which the program would be used. Several of the themes in the next part of this section return to this problem.

On the other hand, adopting these different perspectives did help remove some (though not all) of my prejudices about the old design, and it inspired suggestions for improvement, many of which turned out to be effective. On the more theoretical side, it confirmed the need to consider *distributed models* [diS86] for explaining complex systems where simple explanations fail.

Phase II: Design the New Interface

In the second phase of the experiment, I collected the suggestions that originated in the first phase, organized them into groups of related issues, and responded with changes to the design of BiblioText's interface (section 5 "Redesign"). A few changes were easy to design and implement, but in most cases complications appeared from remarkably diverse sources. Many decisions demanded tradeoffs for which no answer appeared obviously better.

Lesson 4. *Good ideas and suggestions often work at cross purposes.*

There were cases where suggestions motivated by different points of view contradicted one another. The tradeoffs required judgement on apparently unrelated issues, especially certain aspects of the context in which the program would be used (see below).

The general themes of multiple points of view and problems with consistency appear again in the next part of this section.

Lesson 5. *Design decisions depend on assumptions about context of use.*

Some suggestions were sensible in isolation, but conflicted with BiblioText's higher level goals, or with contextual influences (notably UNIX). Other suggestions were perfectly sensible and coherent from the perspective of the program and its users, but required support not available in SunView. Some of the tradeoffs required policy decisions on issues I had not previously considered: presumed user skills, learnability vs. convenience, preferred organizations for *bib* data, and others.

Lesson 6. *Everything is related.*

Even my attempt to categorize suggestions by issue failed. Later experience demonstrated that suggested improvements in one group (editing functionality) were irrelevant to users and would remain so until certain issues in another group (storage formats for the collection) could be resolved.

Phase III: Evaluate the New Interface

In the third phase of the experiment, I collected informal feedback from two new users (section 6 "An Empirical Evaluation") and reviewed the success of the design changes I had made. (section 7 "The New Interface Reconsidered").

Lesson 7. *Utility is strongly driven by working context.*

The experimental subjects were the first users of BiblioText other than myself. I was startled by the divergence of our three working contexts, given the similarity of our situations. Fit between tool and context seemed to dominate perceived benefit from the program. One of the subjects will continue to use BiblioText and the other will not; the quality of the interface and even details of functionality were not major factors in their decisions.

Lesson 8. *Low level details dominate.*

Most of the subjects' immediate difficulty concerned points of interaction, in particular dealing with the interactive behavior of SunView screen objects. I had to conclude that (at least for new users like these two) low level details of interaction dominate issues of functionality, and that by the criterion of unobtrusiveness, BiblioText's user interface is still deficient.

Lesson 9. *It is easier to evaluate learnability than utility.*

It is difficult to judge the relevance of these results to the goal of making BiblioText productive and convenient. Like so many studies in the literature, the results of the empirical evaluation are biased toward the view of new users. For example, a regular user of the program would eventually cease to be surprised by the behavior of buttons, menus, the like. Some of the subjects' problems resulted from incomplete understanding of the program's functionality, but were easily resolved with a short explanation, for example by appealing to advanced knowledge of *bib*. Even though the particular functionality is idiosyncratic, it would cease to be a problem to these users.

8.3. Themes

More general issues emerged throughout the experiment; the following comments discuss some of these recurring themes.

Specialization vs. Generality

Two common themes in the literature on user interface design are (a) clean structural models, directly presented to the user, and (b) simple mappings between those models and the user's task domain (the work to be done). In practice, these can work at cross purposes and can be confounded by other issues.

An instance of the conflict is the tension in any design between specialization and generality. One path to simple design is context independence and thus more general applicability. For example a simple text editor with few commands is equally useful to many people. Direct manipulation interfaces strive for this kind of simplicity. But in the attempt to simplify the mapping between a program's structure and a user's (presumably) rich task domain, the designer often adds special functionality to match intended patterns; this kind of functionality is inherently context dependent. Often the most useful of these specialized functions have the least to do with direct manipulation; for example the "Sort" function in BiblioText is one of the easiest to understand, but is one of the least direct and the most difficult to describe formally.

The situation is even more complex when one addresses the needs of multiple users, since their tasks presumably vary and (equally as important) so do their ways of thinking about them. Context dependent specialization reduces utility to users who do not share a common context.

The delicate balance between specialization and generality was upset in many ways during this experiment. For example, I unwittingly overspecialized with design decisions that depended implicitly on the way I organize my *bib* data. This is the familiar myopia of the designer; one never really knows how one's creation will be used.

On the other hand, I overgeneralized with the decision to simplify BiblioText's model by making it more like a cut and paste editor. This change failed because it was not grounded in any working context; the experimental subjects couldn't imagine what to do with it.

Models and Abstractions

My failure to complete a full functional description of BiblioText demonstrates the very reason we build models. I retreated from an avalanche of detail that threatened to obscure the information I sought; I narrowed my view of the program, choosing to think of its functionality without interface. Of course the description I originally started to write would also be an abstraction; it just turned out to be the wrong one for the job. Later parts of the experiment required different abstract views of the program.

I was also obliged by the program's context to adopt particular views that I wouldn't have chosen otherwise. For example, building an interactive SunView program demands that one cast the program, at least partially, into the SunView model.

I adopted yet other points of view for pedagogical reasons, for example "this function mimics the referencing behavior of *bib*."

Multiple Points Of View

Models are inherently incomplete, and we try to understand systems by using many of them. Each model (or description) of BiblioText revealed new aspects of the program, but only by selectively ignoring others.

It would appear that systems like BiblioText are of necessity built that way, not just apprehended after the fact. Weinberg's "superobserver," if such a person could be found, would apprehend the system in its entirety, from all points of view. But as Weinberg himself comments,

> Combinatorial growth is a critical flaw in any discussion of multiple points of view, for though we can imagine that a superobserver might exist in simple situations, there is little chance of having one in situations of even modest complexity [Wei75].

A serious problem with multiple views is that the models they produce can never be completely disjoint (otherwise they would be describing disjoint systems). To a maddening degree, most suggestions for BiblioText's improvement were motivated in some points of view but conflicted with suggestions derived from others.

Formal methods

We rely heavily on multiple models (or abstractions) to design, build, and talk about complex systems; computer programs are no exception. This approach is evident in top-down design, separation of concerns, layers, dataflow, hierarchical decomposition, information hiding, and many more.

There is no free lunch, however, and we often forget the crucial limitation. Any observed fact about a system, based on a particular point of view, does not describe the system; it only describes the model of the system produced by that particular point of view.

This limitation applies to formal specifications of programs. No specification can be more than a partial description, limited by the language (or point of view) in which it is expressed. Even more discouraging, any observation, claim, or proof about a program is only valid for that model of the system implied by the point of view in which it is made. This observation is fundamental to Smith's "Limits of Correctness" [Smi85].

Consider, for example, mathematical proofs of programs that assume real arithmetic (which does not exist in the hardware), termination proofs that ignore interrupts and crashes (which do exist in most hardware), and attempts to specify the "style" of interactive programs (a real, but relative concept). To paraphrase Weinberg:

> Formal specification methods study those systems for which the approximations of formal specifications work successfully.

The same limitation applies when we replace "formal specifications" with "conceptual models". Experience with BiblioText repeatedly demonstrates that details outside a particular point of view can frustrate and dominate results obtained from within. This argues that we interpret the results of research in this area with caution; they yield insight about real systems only in the sense that the "frictionless planes" of physics can help mechanical engineers.

Consistency is a Relation

Discussions of human-computer interfaces often contain careless reference to *consistency*, as though it were an attribute that a system can either have or not have (for *consistency* substitute *uniform*, *friendly*, *transparent*, or your favorite adjective, the point is the same).

Gregory Bateson would call this fallacy an error of logical type, claiming instead that these words describe *relationships* between system and user, and furthermore that these relationships are dynamic [Bat79]. The research literature in cognitive approaches to system design acknowledges the problem by postulating cognitive models that exist in the mind of the beholder, apart from system or designer. It is in the various mappings that relevant properties emerge.

Experience with BiblioText suggests that we would more profitably view these properties as three-way relations among observer, object, and view (more specifically: user, system, and task). Seen this way, it is all but impossible to maintain any absolute sense of consistency.

The subjects of BiblioText's empirical evaluation established immediately divergent patterns of use; each built a personal relationship with the system. The patterns differed from one another and from what I, the designer had envisioned. Furthermore, the subjects' qualitative assessments of BiblioText (and its interface) changed from moment to moment, depending on the topic of conversation, the task at hand, and how much they had learned about the program so far.

8.4. Final Thoughts

Although many of the conclusions I've drawn from the experiment suggest the enormous difficulty inherent in applying any of these approaches to real systems, I don't believe that these approaches are without value. I would no more argue that they should be abandoned than I would argue that physics teachers should abandon frictionless planes or that programming teachers should abandon stepwise refinement.

I believe, however, that we will not see the time when cognitive approaches alone can guide us to a successful computer system design. Instead, they will increasingly (I hope) offer the kinds of benefit that they did during this experiment: a richer vocabulary of interaction, heightened sensitivity to contextual diversity, varied frameworks for asking questions, and deepened insight into the problems of interface design.

9. Acknowledgements

Robert Ballance supplied helpful suggestions for the original prototype design. Eduardo Pelegri-Llopart and Dain Samples helped evaluate the user interface of an intermediate version.

Profs. Andy diSessa, Susan Graham, Mike Harrison, Peter Pirolli, and Larry Rowe all helped create the academic setting in which this research could take place.

10. References

[Bat79] G. Bateson, *Mind and Nature: A Necessary Unity*, Bantam, 1979.

[Bro88] F. P. Brooks, Grasping Reality Through Illusion — Interactive Graphics Serving Science, *Proceedings SIGCHI Conference on Human Factors in Computing Systems*, Washington, DC, May 1988, 1-11.

[BuL82] T. A. Budd and G. M. Levin, A UNIX Bibliographic Database Facility, University of Arizona Technical Report 82-1, 1982.

[CaT82] J. M. Carroll and J. C. Thomas, Metaphor and the Cognitive Representation of Computing Systems, *IEEE Transactions on Systems, Man, and Information Sciences SMC-12*, 2 (1982), 107-116.

[diS86] A. A. diSessa, Models of Computation, in *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. A. Norman and S. W. Draper (editor), Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, 210-218.

[Fel79] S. I. Feldman, Make — A Program for Maintaining Computer Programs, *Software—Practice & Experience 9*, 3 (March 1979), 255-265.

[Get86] J. Gettys, Problems Implementing Window Systems in UNIX, *Proceedings Winter USENIX Technical Conference*, January 1986, 89-97.

[Goo87] D. Goodman, *The Complete HyperCard Handbook*, Bantam Books, New York, 1987.

[HaM82] F. Halasz and T. P. Moran, Analogy Considered Harmful, *Proceedings of the Conference on Human Factors in Computing Systems*, Geithersburg, MD, March 1982.

[HMT87] F. G. Halasz, T. P. Moran and R. H. Trigg, NoteCards in a Nutshell, *Proceedings SIGCHI Conference on Human Factors in Computing Systems*, Toronto, Canada, April 1987, 45-52.

[HHN86] E. L. Hutchins, J. D. Hollan and D. A. Norman, Direct Manipulation Interfaces, in *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. A. Norman and S. W. Draper (editor), Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, 87-124.

[Joy79] W. Joy, *An Introduction to Display Editing with Vi*, Computer Science Division, EECS, University of California, Berkeley, March 1979.

[Mal82] T. W. Malone, How Do People Organize Their Desks? Implications for the Design of Office Information Systems, *Proceedings SIGOA Conference on Office Information Systems*, Philadelphia, PA, June 1982, 47-49. (Extended Abstract).

[Nor86] D. A. Norman, Cognitive Engineering, in *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. A. Norman and S. W. Draper (editor), Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, 31-61.

[Oss76] J. F. Ossanna, *NROFF/TROFF User's Manual*, Bell Laboratories, Murray Hill, NJ, 1976.

[Smi85] B. C. Smith, Limits of Correctness in Computers, CSLI-85-36, Center for the Study of Language and Information, Stanford, CA, October 1985. Prepared for and presented at a Symposium on Unintentional Nuclear War at the Fifth Congress of the International Physicians for the Prevention of Nuclear War, Budapest, Hungary, June 28-July 1, 1985..

[Sta81] R. M. Stallman, EMACS: The Extensible, Customizable, Self-Documenting Display Editor, *Proceedings of the ACM-SIGPLAN SIGOA Symposium on Text Manipulation*, Portland, OR, June 1981, 147-156.

[Sti80] V. Stibic, *Personal Documentation for Professionals: Means and Methods*, North-Holland, New York, NY, Amsterdam, 1980.

[SSS84] *SunWindows Programmer's Guide: An Introduction to the Sun Window System*, Sun Microsystems, Inc., Mountain View, CA, 1984. Release 1.1.

[SSS86] *SunView Programmer's Guide*, Sun Microsystems, Inc., Mountain View, CA, October 1986. Revision A (3.2 release).

[Van88] M. L. Van De Vanter, BiblioText: A Hypertext Browser for Bibliographic Data and Notes, 88/455, Computer Science Division, EECS, University of California, Berkeley, October 25, 1988.

[Wei75] G. Weinberg, *An Introduction to General Systems Thinking*, John Wiley & Sons, New York, NY, 1975.

[You81] R. M. Young, The machine inside the machine: users' models of pocket calculators, *International Journal of Man-Machine Studies 15* (1981), 51-85.

Appendix A. Original Task Analysis

BiblioText was originally created as part of an earlier project. The design of the first prototype was derived from analysis of a particular working domain: professionals, working singly or in small groups, who need to manage and selectively share personal information including bibliographic references, on-line documents, personal notes, and their mutual interconnections.

Analysis of the selected working domain began with a list of tasks that a person might want to perform, along with a discussion of how to accomplish those tasks in the absence of any automated tools. This appendix presents that discussion, since it is the background for part of the experiment described in the body of this report (section 4.2 "Task Analysis").

The Task Environment

Ignoring automated aids for the moment, the task environment in which the project was situated concerns the needs of a small group of people (perhaps only one person) who maintain (and share) documents, in particular more documents than can conveniently be kept in view or in mind. The term *document* in this context refers to books, articles, reports, and notes, but also possibly to microform, audio and computer tapes, transparencies, or anything else of sufficient interest to the parties concerned.

This working environment falls between two extremes in size. On one hand, a person who uses few documents, or who seldom retrieves previously read documents, need not address issues of management, sharing, indexing, retrieval, and browsing. On the other hand, general libraries, even of medium size, present problems that are distinct from those discussed here.

Typical Tasks

This list summarizes the originally proposed tasks, all concerning the maintenance, augmentation, and perusal of a personal (or shared) information repository. These tasks are described independently of any automation. The descriptions here note interrelationships among the tasks with references to other tasks in the list, such as (a).

(a) *Notice an interesting document.*

One may hope to read the document immediately (b), but it will often be necessary to put it aside until some later time. The ubiquitous solution is to add it to some "IN" box or *pile* where it waits and serves as a visual reminder [Mal82]; this can, of course, create retrieval problems when a document needs to be in more than one pile at a time, or (when borrowed) in more than one office (d). Less effectively, a written note can serve as a reminder and pointer.

(b) *Read an interesting document, possibly write notes.*

The notes must be put where they can be retrieved. If not stored with the document itself, the notes must contain some pointer to the document, an informal citation. The document itself may be put with other, related documents (h) if another use seems imminent. Otherwise the document is returned (if borrowed) or filed (c).

(c) *Acquire an interesting document.*

Whether one reads it (b) or does not (a), the document must reside somewhere. One typically stores documents so that their physical arrangement is of some help for later retrieval (f). They may be *categorized* in some useful way, but often the categories have as much to do with physical characteristics as with their content (one tends not to store books and article reprints together). Ambitious users may *index* new acquisitions to expedite later retrieval. In any case, one must have some hope of locating the new document in some future context where it might be relevant. Stibic [Sti80] describes manual techniques by which the individual professional may conveniently categorize and index documents.

(d) *Lend a document.*

Since most retrieval schemes are based on location, the owner must somehow record the loan. The alternatives are wasted time and shrinking libraries.

(e) *Locate notes on a specific document.*

Retrieval depends on storage and organization. Notes may be in folders or piles organized by subject; they may be organized according to the context in which the documents were originally read (such as a class reader or bound tutorial); or they may be organized in parallel with the documents themselves (stored in folders along with reprints or stuffed between the pages of books). If the notes reside in computer files, it may be necessary to produce a printed version.

(f) *Locate a specific document.*

For small libraries, simple search may suffice. Larger collections, including the libraries of many professionals, demand additional mechanism. Some documents, such as articles reprinted in bound volumes, can be particularly difficult to locate manually. Closely related tasks are to answer the questions "do I possess some specific document?" and "who has my copy of this document?".

(g) *Write a document, citing other documents.*

The first step is to remember which documents are relevant. It also may be necessary to locate and review related notes (e) and source documents (f), and possibly to copy excerpts into the new document. Finally, the author adds formal citations to the new document, based on notes or on the original documents.

(h) *Maintain for later retrieval a collection of related documents.*

Whether one has read them and whether one possesses them, one often wants a collection that represents readings on some specific subject, or associated with some particular project. These tend to grow incrementally and to be reorganized occasionally. The physical piles described under (a) are common; people seem to respond well to the visual cue of the pile. Written reading lists can also be used; unfortunately, a reading list is itself a document and needs some filing/retrieval mechanism. Piles of documents are hard to lose, though in the extreme they can make life uncomfortable.

(i) *Extract a new group of documents according to unanticipated criteria.*

Work often creates new contexts that prompt searches for related documents. Memory suffices up to a point, simply browsing through shelves is effective, asking one's colleagues helps, but a cost-effective indexing mechanism (paid for at acquisition (c)) can make these searches much easier. As Stibic argues [Sti80], it is not cost effective for

the professional to reinvent the elaborate on-line retrieval systems appropriate to institutional libraries. Personal indexing schemes can be adapted to the particular interests and needs of its user(s), and can therefore be much more effective. Finally, it may be important to print a formal or informal list of the results.

(j) *Peruse personal notes, either from a maintained collection or following some unanticipated criteria.*

In either case, retrieval depends on their storage and indexing (b and e). Perusal may consist of thumbing through and rereading, but for more extensive work there seems to be no substitute for the large table top. Often, that organization itself, the final arrangement of notes on the table, is an important product of the information search. If the notes reside in a computer file, it may be necessary to produce a printed version of the collection.