Bookshelf--A Network Online Reference Tool

Overview

by Linda B. Merims

1. Purpose

Bookshelf is designed to provide online access to information in a distributed network environment.

1.1. Athena's Needs

1.1.1. Essential Introductory Documentation

Athena has its own need to get documentation available to users. The present system of placing n handouts in n public clusters has gone as far as it can.

- It is costly to print the material, approaching 50K annually.
- Students read the material through once to learn the system. Subsequently, however, they pick up copies in clusters for quick reference, and then throw them away.
- It takes one student "circuit rider" some 10 hours per week to distribute the items to the clusters.
- It takes a lot of administrative overhead and time to keep paper masters of documents, create revisions, and then print the new documents in time for the start of Fall and Spring term.
- The approximately 21,000 Essentials printed for each term take up a lot of storage space.
- As Athena evolves, public clusters will no longer be where most people do their work and distributing to private sites is unfeasible.

1.1.2. Unix Documentation

Athena's problem is not confined to introductory Essentials. The Unix system manuals are only sold in six large "bundled" volumes, costing about $15 apiece. If one wants only the 30-page Fortran manual, one must buy a 300-page document. This material is available online in */usr/doc*, but as unformatted files. We also have sources to some of the applications software manuals such as Gnu Emacs and X, but, again, these are not easily read online. *Bookshelf* can make this material approachable to the user. They can read it online, print what they want, and only pay for what they need. Athena can eliminate the administrative difficulty of dealing with IS to get manuals printed and available to users.

1.1.3. Limited Interest Documents

Bookshelf can also provide a way to handle the "confetti" problem. Many small odds and ends pieces of useful documentation have been written (for example, the Consulting Seminar series, or the Athena Technical Plan). The audience for these pieces does not justify their being printed and distributed wholesale like Essentials. Nonetheless they *are* useful. *Bookshelf* provides a low overhead way to make this material accessible to users.

1.2. Faculty Developers

Faculty developing software and teaching a course have a wide range of documentation needs that *Bookshelf* can help satisfy. There is the problem of teaching the student how to use the software itself; problem sets have to be distributed, and often the professor wants to distribute supplementary notes and readings.

Athena provides no tools to help faculty do this except mail, "copy this file from this directory," and turnin/pickup. Notice that these are all system-level programs that the student must use as separate steps in addition to the course software. *Bookshelf* will be implemented as a library call that can be included in X applications. Faculty can use the metaphor--which students will already be familiar with from other contexts--to satisfy some of their document distribution needs.

1.3. General Users

Information sharing is one of Athena's oldest and most basic goals, yet it is one we have done almost nothing to realize. Just as Athena is one actor on the network with a pile of material we want the community at large to be able to get at without needing our attention and active effort for every request, so every department and organization has a collection of material they might like to put on a public shelf: course requirements, thesis deadline schedules, summaries of subjects, faculty lists, software documentation, counseling material, recruiting material, and perhaps software.

When the distributed network files system becomes available, people will be able to setup their machines as servers of this information. But how is one to find it? We will be dealing on the level of machine/server names, files and directories. *Bookshelf* can provide a system-wide metaphor and "join up" mechanism that will make it easy to know where to put something if you want to share it, and where to look for something if you want to find it.

2. How It Works

This will be an internalist discussion, so if you havn't read the external description of *Bookshelf*, you should do that first, since this won't give you much flavor of the interaction.

Bookshelf is composed of at least six separate entities, each of which may be developed largely independently of the others, and to a different degree of sophistication. To institute all entities to their fullest is a sizeable project; I have in mind a limited implementation to get us started.

2.1. Bookshelf

Bookshelf is the piece that each person starts out with. It consists of a ***.bookshelf*** directory containing a ***shelflist*** file. The Bookshelf program reads the shelflist and from it

produces the shelf picture with the books on it in appropriate thickness and orientation (the ***displayer***). As the user rolls the mouse over the books, their titles appear. When the user clicks on a mouse, the ***executor*** discovers which program and file to start up from the *shelflist* and causes them to start. Most usually, the program started will be *Bookshelf's* own ***browser***. Bookshelf always guarantees ultimate user control by maintaining an inviolate exist button that the user can press that will close the book.

Bookshelf also contains utilities that let the user ***edit*** the bookshelf, and ***copy*** books between shelves.

2.2. Browser

Browser is the piece that displays the books. The browser is page-oriented. It displays books with shading to indicate thickness and how deep into the volume you are. You can page forward and backwards, and randomly jump to any place you point to. The ***annotator***, ***bookmarker***, and ***printer*** are called by the browser. When indexing information is available, it can be used to hop and about and search the book. Browser supports a variety of book types; initially just text and later formats such as Postscript.

Although Athena will supply a standard browser, a volume may call its own displaying program instead.

2.3. Types of Books

Books can be classified by two axis: the form of the data (ascii text, ascii text plus line printer control, Postscript), and by abstract organization (book, binder, scrapbook, notecard).

2.4. Indexing

Indexing is done within volumes. I have given no thought to indexing across volumes. One can index upon chapter/section/subsection titles, and by words flagged as index words. The basic model is that displayed by the VAXstation-100 Help system of several years past. The indexing information is roduced as a *by-product* of the formatting run: i.e., Scribe, nroff/troff, Tex, whatever, produces the indexing information at the same time it is constructing its table of contents and indexes for printer output.

2.5. Backroom

Backroom is the next higher abstraction up from Bookshelf. Where *Bookshelf* is one shelf, holding about 10 volumes; Backroom is a wall-full of shelves holding 100-300 volumes. Backroom is built out of the same pieces (*displayer, executor, editor*) that Bookshelf is. Perhaps it reaches that level of organization where a shift to database representation is necessary. Backroom icons can sit on Bookshelves, and can be copied as any volume can.

2.6. Union Catalog

Each volume in the system is represented by a catalog card. When you copy books, you are only copying catalog cards around. There is no *necessary* binding at all between a volume and its catalog card. I suspect that, if Bookshelf is going to break, it will break

right here. Cards move around, volumes do too, data ages, and anything left to human tidyness as its sole guarantee of integrity will rapidly go to pot. The ***Union Catalog*** is a central database where one can elect to register one's catalog cards. Volumes will be assigned accession numbers. The Bookshelf program will have background processes that check catalog cards against changes in the Union Catalog when an access fails, or upon request of the user. When the owner of a catalog card changes it, the Bookshelf editor will transfer the information to the Union Catalog, again, in the background. In this way, pointers may be kept properly bound to objects.

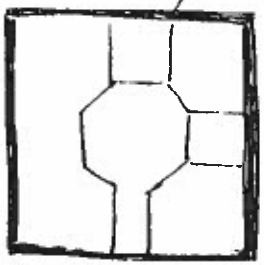
3. A Continuum of Aggregations

Figure showing how *Bookshelf* is the smallest level or organization in a continuum--a "one". *Backroom* represents a three. At the other end of the spectrum is ***Library***, a sophisticated interface capable of handling millions of volumes.

Need 3's & 4's
for personal &
small organization needs
"BACKROOM"

Bookshelf

Library



100-300
volumes

1

5

10

1-10
volumes

Simple

Intricate,
Monolithic

4. Building Out of Common Pieces

Figure showing how the different entities on the continuum are built out of higher aggregations of catalog cards/shelflists/catalogs, and displayed and edited using programs that call the same basic library routines.

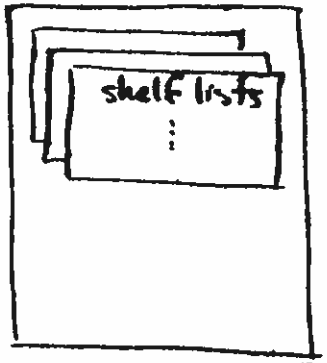
bookshelf

backroom

shelflist

+ shelf display info

catalog card

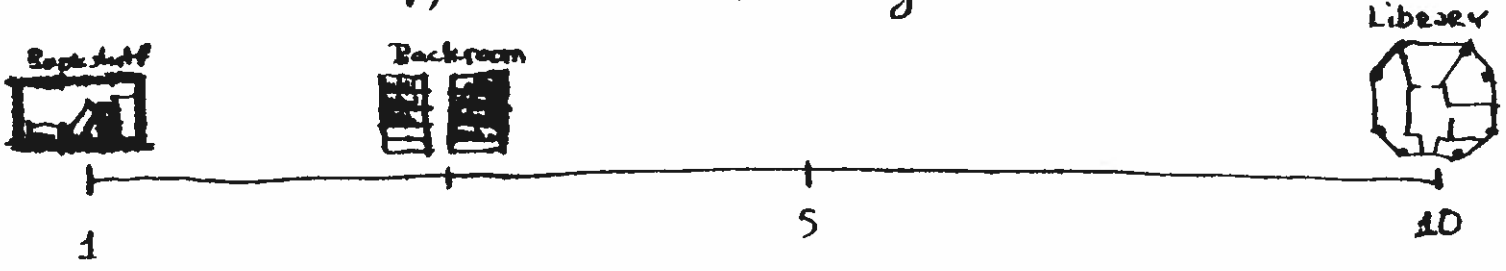


+ display info

* * *

shelflist reader / displayer
- select, exec, close
- data display & edit
- copy — exists independantly

} utilities common to both with only slight modification



5. General Design Truths

- Bookshelf should be exportable. A stripped-down, single machine version should work on straight X and Unix.
- Where the scale of a component is "cottage," it should be implemented with cottage tools. E.g., the single bookshelf shelflist should not be an Ingres database, but some standard Unix component like ndbm. We should not convert to monolithic tools until the scale requires it.
- If a piece does rely on a peculiar software package, that package should be included with the sources, where possible: e.g., *gdb*.
- Bookshelf should not rely on dead-end tools. It must run on both types of Athena hardware.
- Where monolithic or peculiar system facilities must be used, the programs should be coded modularly so that the peculiar piece could be replaced by a different tool somebody else has that would serve the same function. E.g., what database system supports the Union Catalog? What network file system(s) is/are supported? These should be pluggable modules with documented interfaces.
- I dont want the equivalent of a marvelous program coded in Joss on a Cyber 74 that only works on Evans and Sutherland graphics displays. We should take the lesson from Unix and X.
- Each major piece shall have three interfaces:
  - a Unix command
  - a visual interface
  - a library call that works on X

6. Initial Implementation

I am anxious to see a limited functionality prototype very quickly. I would shoot for:

- Defining a limited shelflist, perhaps just a title and local filename.
- A Bookshelf program that:
    - Could create the blank shelf,
    - Interpret the shelflist file to display the volumes. These would not be fancy spines of apprpriate thickness; just blocks.
    - Show titles as you rolled over them,
    - Let you pick a book, the executor enforcing an EXIT button.
- Could start *more* (perhaps hardwired in; I'm not assuming the executor, yet) on a text file.
- Let you print all of the file.
- Returning to Bookshelf, let you copy between bookshelves.

It would only work on local files. It would be a library call. The next thing I'd go for would be a simple editor for the shelf. (We would just emacs a text shelflist at the start).

The intent here is to get something up that, in sketchy form, plays the main parts of the story. It should be like a pencil version of a Disney animated feature. The entire running time of the movie is represented, with as much as one has for each scene standing in its appropriate slot, whether still sketch, live action film, pencil, or completed segment.