The Digital Library Integrated Task Environment (DLITE)

Steve B. Cousins, Andreas Paepcke, Terry Winograd
Stanford University
Computer Science Department
Stanford, CA 94305 USA
+1 415 723 7784
{cousins,paepcke,winograd}@cs.stanford.edu

Eric A. Bier, Ken Pier
Xerox PARC
3333 Coyote Hill Rd.
Palo Alto, CA 94304 USA
+1 415 812 4000
{bier,pier}@parc.xerox.com

ABSTRACT

We describe a case study in the design of a user interface to a digital library. Our design stems from a vision of a library as a channel to the vast array of digital information and document services that are becoming available. Based on published studies of library use and on scenarios, we developed a metaphor called workcenters, which are customized for users' tasks. Due to our scenarios and to prior work in the CHI community, we chose a direct-manipulation realization of the metaphor. Our system, called DLITE, is designed to make it easy for users to interact with many different services while focusing on a task. Users have reacted favorably to the interface design in pilot testing. We conclude by describing our approaches to this problem.

Keywords: Digital library, user interface, direct-manipulation, world-wide web, holophrasting

1. INTRODUCTION

The Stanford Digital Library project is focused on creating technology that will allow a user to access digital resources, from static document citations to dynamic information exploration and management services, without having to know the details of the format of each document and the mechanics of each service. This technology, loosely called the "InfoBus" [Paep96a], provides a unifying framework that can bring together services now provided on the world-wide web, as well as traditional information retrieval and document services, and new kinds of information services that are being developed.

The InfoBus technology makes access to the resources possible, and must be accompanied by an interface that provides a consistent model for users to deal with the plethora of offerings. Now that the web has brought us consumer-level distributed systems, we need powerful metaphors and systems to help us handle the new capabilities. DLITE is the result of a task-oriented, user-centered design process. It is a system that a person can use to interact with many services in pursuit of a complex goal.

A spectrum of simplicity versus power is being played out with the development of the world-wide web. The web began as hypertext: the ontology consisted of pages and links, and the user interface was very simple. Pages contained formatted text and images. Programs could be invoked, but their parameters were limited to those that could be encoded in a link. User interface designers could only give users access to program parameters by setting up a choice between links.

HTML forms changed the web from a tool for browsing to a tool for building distributed user interfaces of the kind found in simple interactive systems. User interface programmers could use a variety of widgets to let users specify how programs would be invoked. The user's conceptual model was more complex, but the ontology still consisted of pages and links, where the notion of a page now incorporated fill-out forms.

The current generation of web browsers gives user interface designers even more control over what users can see and do. Using Java applets, designers can implement the direct-manipulation interfaces that the CHI community has been talking about and using for years. These interfaces allow users to point to objects without having to name them explicitly.

Applet-based systems like Java provide the means to build direct-manipulation interfaces that have the same level of user-interface technology as single-user systems. However, the design of previous GUI systems has been shaped by an environment (the PC or workstation) which has different characteristics from the heterogeneous and distributed environment of the net. Although there is much experience building direct-manipulation interfaces, building direct-manipulation *distributed* interfaces raises new challenges. For example, in our domain we can draw an icon representing a remote service, but in addition to whatever "normal" manipulations the user can perform, we also need to show the state of the network and of the remote service. In this work, we explore how direct-manipulation can be used to build a user interface for a distributed information system.

DLITE is a digital library interface that

- gives the user control in a task context,
- provides smooth integration of services,
- supports services that run at very different speeds, and
- supports sharing, re-use, and persistence.

We are developing the DLITE user interface assuming that eventually we will be able to widely deploy a direct-manipulation, drag-and-drop style interface. There are two versions of the interface: one is implemented using Python and Tk on X Windows, and one in Java's simple windowing toolkit AWT. Both implementations use the Netscape Navi-

gator web browser as an auxiliary input and display mechanism. We use CORBA distributed objects [OMG93] as our distribution framework, as implemented by Xerox PARC's Inter-Language Unification (ILU) system [Cutt93].

In the next section, we describe the workcenter metaphor. Then we briefly discuss the objects that can exist in a workcenter, as background for an extended example of the system in use. We then summarize the interaction modalities of the system, and discuss user reactions from a pilot study. Finally, we discuss some of the issues that surfaced and how DLITE relates to other systems.

2. WORKCENTERS

We introduce the notion of a workcenter, that is inspired by the work on Rooms [Hend86]. A workcenter is a place where the tools for a task are ready-to-hand. A kitchen is a good real-world example of a workcenter: the tools for cooking are handy. You could do woodworking in a kitchen, but a kitchen is not set up for woodworking.

Workcenters suggest activities that can be done in them, just as a kitchen is a subtle reminder of what you might cook for breakfast. In DLITE, the tools are called "components." For example, a DLITE workcenter might contain a component for automated summarization of documents. If such a component were present, it would be because the designer of the workcenter put it there for a reason, or because a previous user of the workcenter had found it useful.

We expect that DLITE workcenters will be created by people with expertise in information exploration and management, such as librarians and editors, and that these expert users will tailor the workcenters to the needs of their colleagues and patrons. For example, a workcenter at a walk-up terminal in a public library might be designed for anonymous users to access free or advertising-supported services. A similar search-oriented workcenter at a research firm might have a component for accessing Knight-Ridder's commercial Dialog information service in a prominent location. A reference librarian's workcenter could have components that are powerful and efficient, but that require training to use effectively. Each user would have access to any number of workcenters, each appropriate for a specific task.

The digital library should support publishing tasks as well as retrieval tasks. A company might have a workcenter for publishing papers, that could include components for routing documents for intellectual property approval and automatically adding approved works to the company's list of publications. As one experiment, we have been building a workcenter to help with the task of producing a high-quality digital document from a printed color document.

Workcenters and their contents persist over time, allowing users to come back and continue a task at a later date. This is consistent with our reading of the library use literature [Marc95, Nard96, Oday93]. Persistence also reinforces the notion that the workcenter is a place, and supports sharing. DLITE is designed so that multiple users can interact with the same workcenter from different physical locations at the same time. This collaboration mechanism will be presented in a later paper.

3. COMPONENTS

All objects in a workcenter are components. DLITE components fall into five basic categories: documents, collections, queries, services, and representations of people.

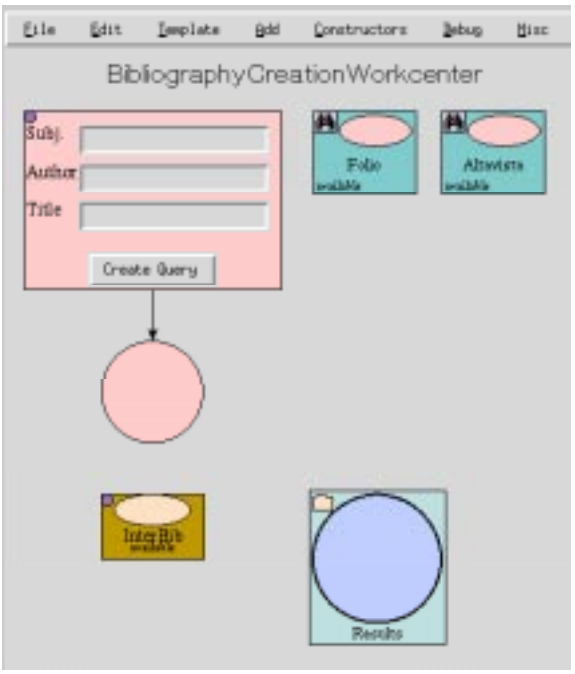

Document components may be anything from simple citations to complex entities with hundreds of meta-information fields and multiple content representations. In our environment, common document types are results from searches of Dialog databases, library bibliographic holdings, or the world-wide web, and documents that users upload from their local disks for processing (such as Microsoft Word documents, text files, or scanned images).


Collections are containers for other components. Basic collections are used to build up sets of interesting information that can be displayed or processed by services. A common sub-type of collection is the result set, which is filled in by search services in the course of processing. Result sets are interesting because they can contain partial results. They begin empty, and a search will put some initial results in them. Users can ask result sets for more results, at which point the underlying InfoBus protocols do the necessary work to reconnect and continue the search.

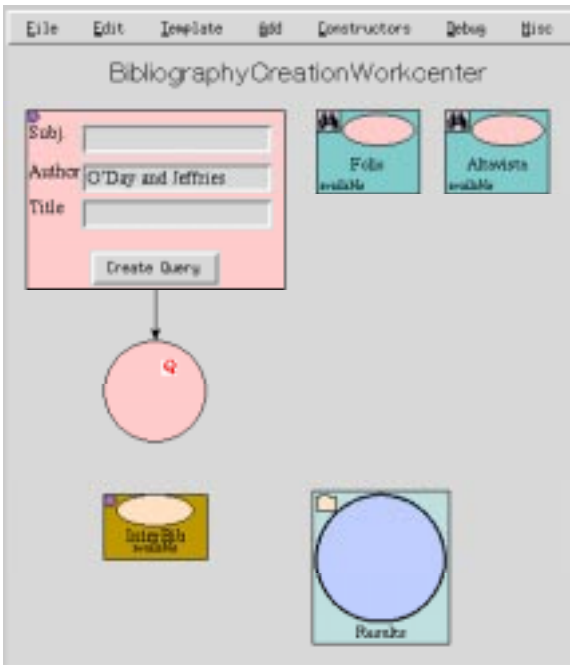

Queries are expressions of a user's information need, and range from simple lists of keywords to complex boolean expressions. The InfoBus has a query translation system built in, so that the same query can be sent to many different search services [Chan96]. From the HCI perspective, the challenge is to help users construct queries that will make sense for the sources in the current workcenter. Queries represent encapsulated retrieval expertise. Since they are first-class objects in DLITE, they can easily be re-used or shared with other people.


The InfoBus contains many different services. There are now hundreds of collections accessible via search services, and other types of services as well, including summarization, optical character recognition, query expansion, format translation and bibliography processing. These services are commonly implemented by building proxy objects that translate from the native protocol of the service to the InfoBus protocol that we use to integrate service components into the user interface. This architecture provides great flexibility in bringing services on-line, but because it allows transparent integration of independent services, it also provides potential points of failure that must be accounted for in the user interface.

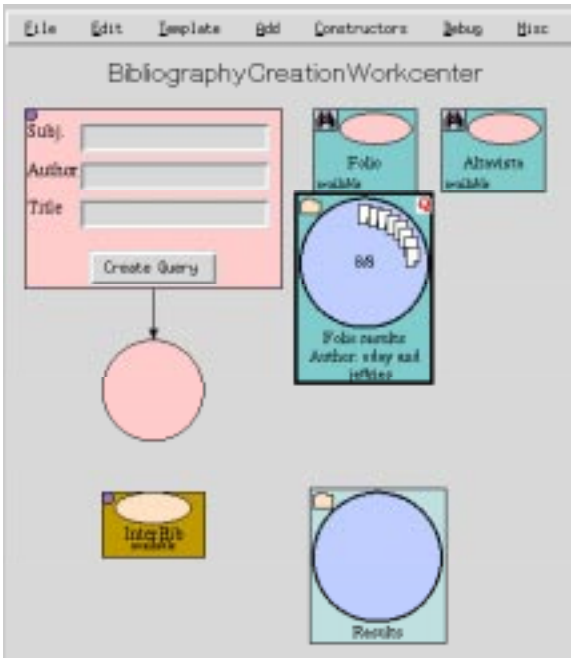

Finally, representations of people in the interface are used to allow the user to manage concerns such as access control, communication, payment, and authorization. For example, a simple icon of a person on the bottom of the workcenter indicates who is currently logged on to the workcenter, and therefore who will be responsible for payment if a query is sent to a fee-based service. The underlying mechanism will automatically send account information when the service is accessed [Cous95].



(a)



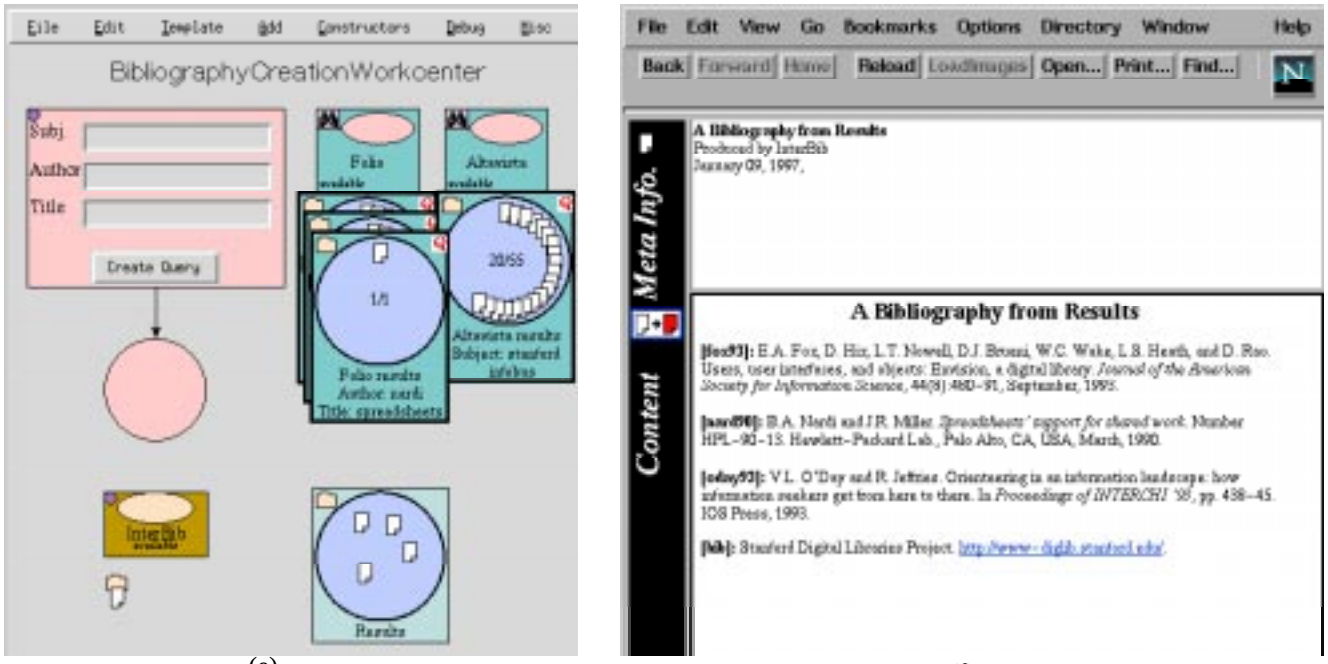
(b)



(c)

(d)

FIGURE 1 (a-d). A DLITE workcenter for bibliography processing. (a) The workcenter template before processing starts. (b) The user creates a query for the authors "O'Day and Jeffries". (c) The user sends the query to the Folio search service by dropping it onto the service. The service responds by creating a result set object, attaching the query to it, and moving the result set away from the search service. As results come from the search service, they are attached to the result set (small document icons). (d) The user activates the collection to see a summary of it in a web browser. The summary contains buttons that can affect objects in the workcenter.



(e) (f)

FIGURE 1 (e-f). (e) The user does more searches, and collects relevant documents in a personal collection. The personal collection is then dropped into InterBib which creates a bibliography object. (f) The resulting bibliography is uniformly formatted in a form suitable for inclusion in a technical paper.

The ontology for the digital library differs from that of a typical desktop with drag-and-drop. Documents differ from files in a number of ways. For example, documents are not necessarily associated with disk storage, and they may be citations to information objects that do not even exist in digital form.

Collections and services also differ from folders and application programs in similar ways. Collection components are visualizations of InfoBus collection objects, which may be stored at a remote computer and accessed over the network. Folders or file directories are usually (but not always) associated with disk storage. Service components may correspond to application programs, but may also be associated with objects running at remote sites. They also contain additional meta-information, in the form of defaults for how the service is to be invoked within this particular task, while applications programs in conventional interfaces are usually customized on a per-user basis at best.

4. LEVERAGING THE WEB BROWSER

We have taken advantage of the fact that web browsers such as the Netscape Navigator can be controlled remotely by designing DLITE to work alongside the browser. In particular, we use the browser to display documents which may have been retrieved from search services, or were generated by other parts of the system. This view includes meta-information, controls that allow documents to be marked in the workcenter, and access to the document content. The marking mechanism is particularly useful because it allows us to keep the visual representation of documents small in the workcenter, saving valuable screen real estate. This integration of a standard Web browser also allows users to move smoothly between searching, browsing, and document processing.

In the next section, we show how the components interact in a walk through of a typical use of one workcenter.

5. EXAMPLE

A common practice when writing research papers is to delay formatting the bibliography until the very end of the process. Researchers often leave comments like "[[cite O'Day&Jeffries]]" in the text, intending to replace them in the final step with a marker like "[1]" and add the appropriate reference at the end of the paper. Tools to help with this task have existed for a long time, such as EndNote, BibTex, and Refer, but they have typically assumed that the researcher had access to a large database of citations in the right format. Researchers would either spend time maintaining and extending files of citations, or would manually format the bibliographies at the end of each paper-writing process.

Recently, InterBib, a new bibliography generating tool, was developed by Paepcke [Paep97]. InterBib exists as a free service on the web. To use it, you fill out an HTML form, upload your citation databases and your incomplete document, and wait for the resulting document to be returned. InterBib differs from its predecessors in that it can accept input in many different formats.

InterBib also has an InfoBus interface, and can accept as input InfoBus collections, such as result sets returned from search services. For example, a query to Dialog via the InfoBus proxy will return a result set that can be passed directly on to InterBib. InterBib can either format the entire result

set as a bibliography, or can use the result set as one of several collections to search while processing the citations in a paper.

InterBib, combined with the ever-growing set of collections accessible via the InfoBus, provides a complete tool for finishing research paper references painlessly. These pieces interact more smoothly in DLITE than in existing HTML interfaces. More generally, for tasks that require the use of multiple services, the direct-manipulation approach gives users more power than a forms-based approach would provide. The advantage of DLITE, in short, is that results of one operation can be directly passed to another operation, without the need to save them, copy them to a clipboard, reformat them, or otherwise do manual processing in between.

The workcenter in Figure 1a is designed for the task of completing bibliographies. The component labelled 'Create Query' is a simple query creation tool. Its field labels correspond to the typical search attributes for this kind of task. The components with oval-shaped input places represent two search services: Folio, a Stanford University Library resource serving the INSPEC bibliographic database, and AltaVista, a general-purpose World-Wide Web search service. The component with the round circle is a user-maintained collection of documents. The last component represents the InterBib service.

To complete the bibliographic task, the user begins by filling in one or more of the fields in the query constructor. For example the authors "O'Day and Jeffries" might be used as a query. Figure 1b shows how clicking the button in the constructor creates a query component labelled "Q". This query component may be dragged across the screen. In our collaborative workspaces it can also be shared with other users.

Figure 1c shows what happens when the user drags the query component onto the oval input place of the Folio service. A result set is created by the system, and the query component is attached to the upper corner of that result set. The entire result set is animated away from its source. As this animation happens, the query is sent to the search service proxy object which uses the InfoBus query translation system to translate the query and forward it in native form to the Folio search service.

As result messages are returned from the search service, they cause information on the display to be updated. Initially, the message "0/8" appears in the result set to let the user know that there are 8 results available. Soon the results begin to appear, and the message changes accordingly. DLITE also adds document icons to the result set as the results arrive. With very fast services, all of the results appear almost instantaneously, while slower services add results one at a time. By default, DLITE initially only requests the first 20 documents to be returned from searches.

Dragging the query from the result set to the AltaVista component's input place begins a similar activity on the AltaVista service. Both searches may occur in parallel.

Next, the user clicks on the result collection to have the results summarized in the Web browser (Figure 1d). In the browser window, the user can mark each relevant document, turning the related document icon in the workcenter dark.

The user copies and drags each selected document into the collection of relevant articles (marked 'Results' in Figure 1). When all citations have been found and placed into the collection, that collection is dropped onto InterBib's input place, and after a bit of processing, the collection and the resulting bibliography move out of InterBib (Figure 1e). Finally, the user views the resulting bibliography in the web browser by clicking on it (Figure 1f), and adds the formatted bibliography to the document. (InterBib can add the bibliography directly into the document, but we have simplified the interaction for the purposes of this example).

If the query did not yield proper results, the user can edit or replace the information in the query constructor to create a new query. For services that accept entire documents as queries for relevance feedback, the user could drop relevant document icons onto the service component input place, although we have not yet implemented any proxies to such services.

Imagine what would be required to accomplish this task using existing tools. Even if there were a web page for searching Folio, and even if it gave its results in a machine-readable format, the user would still have to somehow combine the results from Folio and AltaVista into a single file, and then go to a third web page and upload that file for processing. By contrast, this task is completed in DLITE without requiring the user to change focus: all tools are ready-to-hand.

6. INTERACTION MODALITIES

DLITE affords three basic operations, "point," "activate," and "drag-and-drop." It uses animation to show the user what is happening, and leverages a web browser as a handy document viewer that we assume users can already use. We will describe each of these concepts, and then look at the "result set" component in more detail to make the discussion concrete.

6.1. Pointing

The most basic affordance in DLITE is pointing. When the user moves the mouse over a DLITE object icon, a small yellow box appears with a short description of the object. This is done to reduce the screen real estate required to display objects, and allows us to display up to about 100 objects in a very small area of the screen. For more information about the object, the user can use the activate affordance.

6.2. Activate

In our current prototype, activate is invoked by "double-clicking" or by clicking the right mouse button, and is more general than double-clicking on an icon in a standard desktop interface. The component that receives the activate message can do an arbitrary action, and the component gets information about which part of itself was activated, and can take different actions for different parts.

Components in DLITE contain one or more visible graphical elements. Most components have an icon in their upper-left that represents the component as a whole. The default response to an activate message on that icon is to hide the rest of the component so that only the icon is visible. The default response to an activate message on other parts of the component is to display information about the component in the web browser. For example, activating the body of a collection will display the summary of the collection in the web browser, while activating the folder icon associated with the collection will shrink the representation of the collection on the screen.

6.3. Drag-and-drop

Drag-and-drop dates back at least to the Xerox Star GUI [John89], but most desktop file system interfaces do not exploit the power of the technique. A document may be dropped onto an application program, but the mechanism ends there: once the application program starts, the icons are covered by application windows and not seen again. Some programs process documents directly, so that drag-and-drop is the only user action necessary. For example, some utilities for uncompressing documents work this way. Unfortunately, users are often left guessing where the results went, and have to resort to other clues, such as menu or application bars to see if the program has even finished yet.

We use animation to indicate that the service has begun processing, is finished processing, or has raised an exception. We allow components to be attached to one another. When a subcomponent is dragged, the semantics may be to remove it, copy it, or drag a link. This is a generalization of the situation (with all of the attendant problems) in desktop file systems -- in those systems the only attachment possible involves objects within folders.

Users can use the icon in the upper-left of components to iconify components in the interest of screen real estate. But they can also drag the icon off of the component to pass the component to a service or to make a copy of the component. This affordance is both powerful and potentially very confusing to users, and we are watching reactions to it closely to find ways to provide the functionality in a non-confusing way

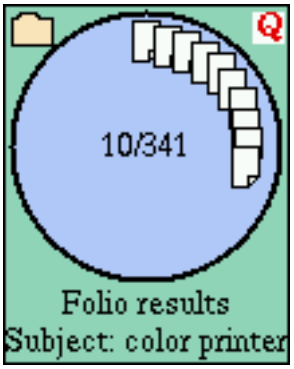
One problem with conventional uses of drag-and-drop is that it requires recipient components to have single inputs. In our context, the services represented by our components often have more than one input parameter. Because we allow intermediate objects (such as queries) to be first-class objects, we can use an iterative technique to solve this problem, by creating objects that have one or more parameters fixed, and can then be provided as parameters to services.

6.4. Animation

Animation is used extensively in DLITE to convey to the user that an action is in progress. When drag-and-drop is used, the object receiving the drop normally moves the dropped object into its geographical center during processing, and then moves it to a standard output location when processing is finished. When an input is dropped where it cannot be handled, it “waggles” by moving back-and-forth quickly, then animates back to the place from which it was dragged.

We have avoided using pop-up dialog boxes since our focus is on conveying as much information as we can in the direct-manipulation style. We are adding more text to the interface on an as-needed basis, as determined by experience with users.

6.5. Example: Result Set Component



A result set is created when a query is dropped onto a search service. This operation provides a good example to illustrate the interaction techniques we have described. A result set contains three types of icons: its folder icon, the query that was used to create it, and a number of documents, which represent the results. When a user points at the icons, descriptions of the objects are displayed in yellow boxes. For example, pointing at the query shows the query text and pointing at a result item shows its title.

Five different activations can be invoked on a result set, depending on where the click occurs. If the user activates the folder icon, the collection is iconified so that only the folder icon is visible. If the query is activated, a view of the query is displayed in the web browser. We intend for this view to support query refinement, but have not implemented this feature yet. If a document icon is activated, the document is viewed in the web browser. If the background of the collection is activated, the entire collection is summarized in the browser. Finally, if the numbers in the center of the result set are activated, the result set requests more documents from the search service. This last activation is a hidden affordance, and is particularly problematic because users often invoke it when they intend to summarize the collection in the browser because of its central location. This particular feature will be changed in the next version of the interface.

All of the icons attached to a result set may be “copied off.” If a user drags a document out of the collection a copy will remain. We tried using “move-off” for the documents, since that is the way that ad hoc collections need to behave, but found that doing so destroyed the integrity of the result collection: once documents were removed from the result set, it no longer represented the results of the query. For the same reason, dropping components onto result sets is not allowed. Using copy- off for the query has proven to be quite helpful. It facilitates query reuse and refinement, when combined with the behavior that queries dropped onto query constructors cause the fields of the constructor to be filled from the query.

Animation is used mainly during result set creation. The current implementation creates a result set on top of the

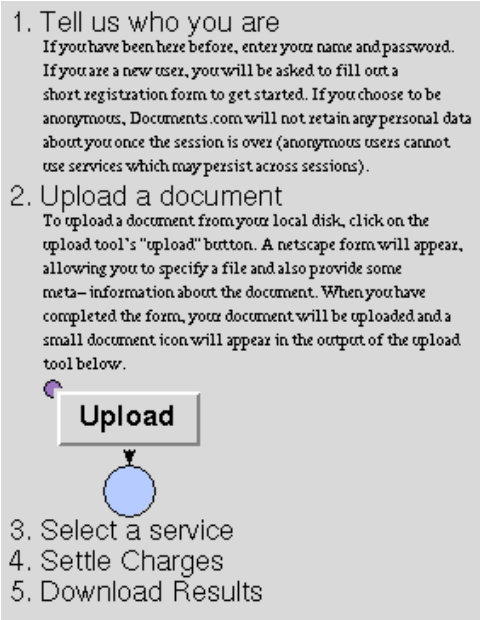


FIGURE 2. A holophrastic interface with items 1 and 2 open.

search service, animates the query to the upper-right of the result set, attaches it, and then animates the entire set away from the service while the search is initiated. As results come in, they are rendered in the result set. The resulting animation, which can only be imagined from a text description, gives the strong impression that the query was accepted and that processing has begun.

7. WALK-UP USER TRAINING APPROACHES

We have explored two techniques for helping users find their way into the DLITE interface: "holophrasting" and "web first."

Holophrasting means the use of a word or phrase to stand for a long text. It was used by the hypertext community in the 1980s to refer to the concept of expanding a link in place (e.g. [Bier92]), but its use in user interfaces dates back to the early 1970's [Hans71]. We have implemented holophrastic components that progressively disclose instructions and other components. The model we have in mind is that a user begins with a short list of instructions summarizing the steps of a task. For each step, the step name is activated to reveal the details of that part of the task. When the task step is finished, it can be collapsed and the next step expanded. Figure 2 shows a holophrasting component for the task of accessing a for-fee document processing service. As users step through these simple instructions, they become familiar with the environment in which the instructions are embedded, and will soon be able to manipulate the environment without the need of such specific instructions.

Since our system is integrated with a web browser, another approach to walk-up users is to assume that they see web pages first. This approach effectively puts the DLITE environment in a conceptual box: the thing that is accessed by clicking on a specific link. It also allows us to ensure that documentation comes first, since it can be accessible from the web instead of being yet another affordance in the interface. Of course, this approach can be combined with holophrasting if both are required or desirable.

8. PILOT STUDY

The DLITE prototype has been under development for two years, undergoing continuous evolution on the basis of feedback from users and observers, both within the project and from outside. An initial pilot usability study was conducted with six novice users.

Each subject was given a copy of a short research paper in which four citations were in the raw, informal shape described in our example above. The subjects were asked to find bibliographic references relevant to the citations, to replace the informal citations in the document with the automatically generated keys for the references they found, and to cause a bibliography to be attached to the document. We gave the subjects a short explanation of the components in the workcenter, and a short demonstration of the affordances. The goal of this pilot study was to learn whether these general explanations were enough to enable users to complete a very specific task with DLITE. We also wanted to learn which aspects of the interface might be confusing.

Our sample consisted of four women and two men, ranging in age from early 20s to late 40s. They were: an engineer with some programming experience but no Internet experience; an administrative associate with no formal computer training but day-to-day experience with Netscape; a teacher with Netscape and computer programming experience; a librarian with extensive training in the use of traditional information retrieval systems; a computer science graduate student; and a government administrator with limited computer experience.

All of the subjects were able to successfully complete the task in under 30 minutes. Because of the small number of subjects, we cannot make any generalizations, but we noted some interesting common actions:

- Two users took advantage of the fact that multiple searches could be performed in parallel.
- Two users clicked on the number of results (asking for more results), when they intended to summarize the collection in the web browser.
- When documents are displayed in the web browser, a document icon is included to remind the user that the page they are seeing originates in DLITE. One user tried repeatedly to drag this icon into the workcenter.
- Two users dragged documents into InterBib instead of collections, and one dragged a folder icon onto a search service, intending to reuse the attached query. In all cases, after wondering why it didn't work, the subjects realized what had happened and continued.
- We have both right-click and double-left-click cause an "activate" event. Unfortunately, Netscape uses single left-click on links to essentially activate them. As a

result, two users repeatedly clicked with the wrong button in one application or the other and waited for something to happen.

We did not test whether subjects will remember the novel affordances of DLITE in this study, but plan to test this in the future.

9. DISCUSSION

Our preliminary results show DLITE to be particularly useful in providing user-level uniformity in interacting with heterogeneous services, and in allowing intuitively obvious parallel activities when interacting with Digital Library services. The latter is important for user productivity in that it matches the multi-track working style required of many Digital Library users [Paepcke96b]. Parallelism all the way up to the user surface is important because different services take significantly different amounts of time to complete.

Here are some design considerations and lessons from the pilot study pertaining to particular aspects of DLITE.

9.1. Workcenters

The notion of workcenter is necessary to partition the space of available resources. There are just too many resources in Digital Libraries that span the Internet to present to the user at one time. Equally important, many of the services will only be useful for particular tasks and would consume valuable screen space and user attention if displayed all the time.

An alternative design for managing the large number of services might be to use a fisheye view technique. The advantage would be that services would at least be in the peripheral view of the user, rather than not being shown at all unless explicitly placed in a workcenter. We decided on our design for two reasons: First, we expect that even for systems that are easy to use, organizations will wish to develop standard collections of tools for given tasks in order to help less sophisticated users, and to ensure that resources are properly used for the organization's tasks. While workcenters could be designed by individual users, they could also be provided by departments such as an organization's research library, or other centers of expertise. This would not be the case with the fisheye view approach.

A second reason for our workcenters is that they make part of the expertise for each task a discrete entity in the system. The collection of tools that are combined in one workcenter may be carefully thought out, and may then be re-used by many users. The fact that a component is included reminds the user of its existence and makes it ready-to-hand. A well-designed workcenter may even suggest an order of activity by the placement of its components.

9.2. Components

One big challenge around our components was to find a small enough set to preserve simplicity, yet to introduce enough of them to span the space of services and notions we needed users to gain access to. We arrived at the present set through a straight-forward requirements analysis. While the set presented here has taken us far, we may have to add a few more components carefully, as the Digital Library grows richer in services. But our strong preference is to keep the number of different component types to a minimum.

The extreme simplicity of our components is both an asset and a liability. We have worked hard to keep the number of inputs to our components to a minimum. In general we do this by allowing users to construct first-class objects (such as queries) out of parts, and then to use those objects as inputs to services. This has proved understandable and useful for the limited examples we have explored, but may need to be revisited when the sophistication of available services (and users' knowledge of them) calls for a less minimalist approach (e.g., in allowing a user to specify more details of the resources to be used in carrying out a particular search, such as time, cost, user context, etc.).

One difficulty for any interface that helps users deal with large numbers of items is the maintenance of display orderliness. One example of this problem is in our decision to have only a few distinct component types. This reduces confusion, but means that there will be many identical-looking objects on the screen. As an alternative, we could introduce more distinctions among the types (e.g., different images for different kinds of documents), increasing information and also increasing visual clutter. Currently we provide simple means for color-marking individual items, and for low-overhead label display (just moving the cursor over the item). Future experiments will determine if these are sufficient.

The one-to-one correspondence between visible components and remote services provides a simple way to distinguish error sources, using the component as an attachment point for the error indicator (which in some cases is a simple as a red X across the component). These indicators can be updated continuously while the user is attending to other processes.

9.3. Animation

In animating the components in the DLITE interface we make it possible for users to track the results of actions and have their attention drawn to salient events (such as a document being added to a result set). In order to avoid confusion, we have found it important not to animate components over long distances on the screen. The animation required significant engineering effort [Cous96] to support our design goal of allowing users to simultaneously share a DLITE workcenter on different machines, potentially separated over wide-area networks.

9.4. Use of the Web Browser

As exemplified by the mouse button issue in our pilot study findings, differences between the Web browser and DLITE interaction conventions sometimes cause problems. It is difficult to provide completely consistent use of the mechanisms, since other interfaces (such as web browsers) have different kinds of functionality. The same problem arises, for example, in ordinary GUIs, where single clicking on a desktop document will select (rather than opening) it, which is not consistent with single-clicking on a link in a browser. We are exploring alternative conventions for selecting, activating, and dragging DLITE component objects, and in our usability studies we will be examining the ways in which

users understand these as integrating with activities in the browser and operating system.

The use of a commercial Web browser as companion to DLITE has otherwise proven to be very advantageous. The browser, of course, covers navigation needs on the Web, and users are often familiar with its facilities. In our Java implementation we are also using the Web browser to deliver DLITE itself.

9.5. Scalability

Scalability of the interface is, as always, an issue. Graphical facilities like DLITE inherently suffer from potential screen real-estate problems. Scrolling is one solution, but it feels more clumsy than when text is scrolled. We have used two mechanisms to deal with scalability strains. One is the use of the Web browser as a surface on which to summarize entire result collections of arbitrary size. While the result collection components are limited as to the number of document icons they can hold, the collection summaries are linearly appended extracts that may be scrolled naturally in the browser. We mostly use document attributes like author, title, abstract and URL for our summaries, but large degrees of sophistication could be used to improve on this simple scheme. The point is that collection summarization does scale.

We are also developing special tools for dealing with large result sets which would be unwieldy in any interface. These tools work through clustering of large result sets which decreases the space needed for display. See [Bald96] for details.

10. RELATED WORK

In the last 15 years, there have been other attempts to give users direct-manipulation interfaces to multiple services within an information system. The Xerox Star and its successors allow simple document objects to be dropped onto applications programs or simple services like printers [John89]. In the 1980s, Metaphor Computer developed “capsules” that gave users the ability to link multiple application programs together. Rose has experimented with extending the traditional desktop with a ‘pile’ metaphor [Rose93].

The Virtual Notebook System [Gorr94], developed in the late 1980s, gave users the ability to combine text and images from disparate sources into a new, shared notebook artifact. Shipman, Marshall, and Moran have experimented extensively with systems for manipulating and understanding information objects arranged on canvases (e.g. VIKI [Ship95]).

Rao’s InfoGrid system emphasized document space visualization and management [Rao92], and Mackinlay’s Butterfly experimented with handling asynchronous requests to services that could take varying amounts of time to respond [Mack95].

Chang and Ungar experimented with animation techniques in the user interface for the Self programming language [Chan93]. Maloney and Smith also incorporate extensive animation in the Morphic user interface [Malo95].

Recently, Hendry and Harper described a system called SketchTrieve that reifies documents and services [Hend96]. SketchTrieve uses much more of a dataflow flavor in its interactions than DLITE, and does not emphasize the variety of services. We will compare the systems in detail elsewhere.

11. CONCLUSIONS

DLITE goes beyond existing direct-manipulation interfaces, bringing distribution of services, asynchronous processing, and multiple representations into the interface in a unified way. Using the metaphor of workcenters containing components, we have experimented with affordances such as drag-and-drop, and feedback mechanisms such as animation, in the context of a functioning digital library prototype. The resulting design supports users in coherent tasks, allowing multiple services to be invoked in parallel and for their results to be smoothly integrated into the user’s environment.

12. ACKNOWLEDGEMENTS

Thanks to Michelle Baldonado for comments on an early draft of this paper, and to Scott Hassan and Tom Schirmer for help with InfoBus integration issues. Alan Steremberg implemented the Java/AWT interface. This material is based upon work supported by the National Science Foundation under Cooperative Agreement IRI-9411306. Funding for this cooperative agreement is also provided by DARPA, NASA, and the industrial partners of the Stanford Digital Libraries Project. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the other sponsors.

REFERENCES

[Bald96] Baldonado, M.Q.W. and Winograd, T. SenseMaker: An Information-Exploration Interface Supporting the Contextual Evolution of a User’s Interests. To appear in CHI’97. Available as http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0048.

[Bier92] Bier, E.A. EmbeddedButtons: supporting buttons in documents. ACM Transactions on Information Systems (Oct. 1992) vol.10, no.4, p. 381-407.

[Chan93] Chang, B.-W. and Ungar, D. Animation: from cartoons to the user interface. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST’93). ACM Press, 1993. p. 45-55.

[Chan96] Chang, K.C-C., et al. Boolean Query Mapping Across Heterogeneous Information Sources. IEEE Transactions on Knowledge and Database Engineering, v. 8, n. 4, August, 1996, 515-512.

[Cous95] Cousins, S.B., et. al. InterPay: managing multiple payment mechanisms in digital libraries. In Proceedings of Digital Library, (Austin,TX, June,1995) http://csdl.tamu.edu/DL95/papers/cousins/cousins.html.

[Cous96] Cousins, S.B., et. al. A Distributed Interface for the Digital Library. Stanford University, 1996. http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0037.

[Cutt93] Cutting, D., et. al. ILU Manual. Xerox PARC. Dec.,1993. ftp://ftp.parc.xerox.com/pub/ilu/ilu.html.

[Gorr94] Gorry, G. A., et al. Experience with the Virtual Notebook System: Abstraction in Hypertext. In Proceedings of CSCW (Chapel Hill, NC, October, 1994), ACM Press, 133-143.

[Hans71] Hansen, W.J. User engineering principles for interactive systems. In Proceedings of Fall Joint Computer Conference, (Las Vegas, November, 1971), AFIPS Press, 523-532.

[Hend86] Henderson, D. A. Rooms: multiple virtual workspaces. In ACM Transactions On Graphics, July, 1986.

[Hend96] Hendry, D. G., and Harper, D. J. An architecture for implementing extensible-seeking environs. Proceedings of SIGIR (Zurich, 1996), 94-100.

[John89] Johnson, J., et. al. The Xerox Star: a retrospective. Computer (1989), 22(9):11-26, 28-29.

[Mack95] Mackinlay, J., et. al. An organic user interface for searching citation links. In Proceedings of SIGCHI, (Denver, CO, May, 1995), Addison Wesley, 67-73.

[Malo95] Maloney, J.H. and Smith, R.B. Directness and liveness in the Morphic user interface construction environment. Proceedings of UIST (1995). ACM Press, 21-28.

[Marc95] Marchionini, G. Information seeking in electronic environments, Cambridge Univ. Press, 1995.

[Nard96] Nardi, B., and O'Day, V. Intelligent agents: What we learned in the library. In Libri, v. 46, n. 2, 1996.

[OMG93] Object Management Group. The Common Object Request Broker: Architecture and Specification. December, 1993. http://www.omg.org.

[Oday93] O'Day, V.L. and Jeffries, R. Orienteering in an information landscape: how information seekers get from here to there. In Proceedings of INTERCHI '93, (Amsterdam, NL, May 1993), IOS Press, 438-445.

[Paep96a] Paepcke, A., et. al. Towards Interoperability in Digital Libraries: Overview and Selected Highlights of the Stanford Digital Library Project. IEEE Computer Magazine, May, 1996. Also: http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1995-0013.

[Paep96b] Paepcke, A. Information needs in technical work settings and their implications for the design of computer tools. Computer Supported Cooperative Work (1996), v. 5, n. 1, 63-92.

[Paep97] Paepcke, A. InterBib: Bibliography-related services. http://www-interbib.stanford.edu/~testbed/interbib. 1997.

[Rao92] Rao, R., et. al. The Information Grid: a framework for information-retrieval centered applications. In Proceedings of UIST (Nov. 15-18, 1992), 23-32.

[Rose93] Rose, D.E. Mander, R. Oren, T. Ponceleon, D.B. Salomon, G. Wong, Y.Y. Content awareness in a file system interface: implementing the 'pile' metaphor for organizing information. Proceedings of SIGIR (1993), 260-269.

.