ELECTRONIC PUBLISHING, VOL . 8(1), 31–48 (MARCH 1995)

A process approach for providing hypermedia services to existing, non-hypermedia applications

CHARLES J. KACMAR

Department of Computer Science
Florida State University
203 Love, Mail Stop 4019,
Tallahassee, Florida 32306-4019
USA

email: kacmar@cs.fsu.edu

SUMMARY

Hypermedia has initiated an explosion of application development based on a navigational model of information access. However, many thousands of existing applications continue to operate successfully without hypermedia services, even if they could benefit from a navigational paradigm. One of the reasons why developers of an existing application might choose to ignore the benefits of hypermedia is the cost of converting the application. If this cost could be minimized, would developers convert to or experiment with hypermedia? What is the cost of conversion? How would a conversion impact the structure or data management components of the application?

This paper discusses these issues from a developer's perspective by presenting three methods of retrofitting existing, non-hypermedia applications to provide hypermedia services. Two methods are based on traditional hypermedia data models and architectures. The third method, and focus of the paper, is an approach that is based on a process model of hypermedia. This approach allows developers to experiment with hypermedia as an information access paradigm without incurring the costs of a full conversion. Moreover, this approach establishes an open environment, leading to application integration under a common framework and allowing any application to participate. The basis of this approach is an autonomous process that is completely external to the application. The facility monitors application activity and provides first-generation hypermedia services to all or selected applications running on a user's display. Thus, even though an application provides no hypermedia services itself, is not aware of and does not depend on a hypermedia model, it can operate as a first-generation hypermedia application as a result of this facility. Implementation details, benefits, and limitations of this approach are discussed.

KEY WORDS hypertext; hypermedia; retrofit; conversion; migration

1 INTRODUCTION

The popularity and use of hypermedia is a partial tribute to the usefulness and success of the paradigm. It is, however, a common fact that many thousands of applications are not based on a hypermedia model nor do they use a navigational paradigm to support information access. Of these applications, some may benefit from hypermedia, some may not. For those that may benefit, what is the cost of converting them to hypermedia and how will such a conversion impact the structure and data management components of the

CCC 0894–3982/95/010031–18
Received 22 July 1994
Revised 24 January 1995

application? If a conversion is performed, will the benefits be as great as expected? How will users respond to the changes in application behavior?

Answers to some of these questions can be achieved through paper and pencil activities that are modeled on hypermedia. Other answers will require deployment of a hypermedia system within the user community. Prototype systems are an effective but interim method of evaluating the use of hypermedia because they incur development costs and may not reflect the full capabilities of the real application.

This paper discusses three methods of migrating (retrofitting) existing non-hypermedia applications to provide hypermedia services. The incentive for this research is provided by Puttress and Guimaraes [1] who advocate a seamless integration without requiring major changes to the existing application.

The difficulty of converting an existing application to support a navigational paradigm is dependent on the architecture of the existing application and the level of hypermedia services that are needed. As we will show, if the architecture of the application is based on a separable [2] or functional [3] interface model, the application can be retrofitted by enhancing the interface component and augmenting it either with an internal or external hypermedia engine [4–6]. If, however, the components of the application are tightly interconnected or the services needed are highly complex, converting the application will require extensive modification of several components. On the other hand, if hypermedia functionality can be offloaded from the application to an external autonomous hypermedia facility, retrofitting can be achieved with minimal modification of the application. This approach, and focus of the paper, results in the ability to retrofit applications easily and provides for an open, integrated hypermedia environment.

To illustrate the issues and importance of a retrofitting approach to hypermedia, consider a graphical application that allows a user to browse through a sequence of scanned images, such as maps or pages of a book. When the application is launched, the first image appears. A user progresses through the pages in a forward or backward direction by pressing either the ‘Next Page’ or ‘Previous Page’ buttons. Suppose the developers and users decide that the application *might* be improved by allowing pages to be related through links. That is, in the case of a collection of map images, this would allow a user to navigate from a ‘high level’ map to subsequent and more detailed maps. In the case of scanned images of book pages, it would allow a user to navigate across pages to reference material or related passages. The question becomes, what is required to add this functionality to this application? More importantly, if the functionality is added to the application but later it is determined that this functionality is *not* valuable, what is the cost of removing it?

The remaining sections of this paper discuss three methods of migrating existing, non-hypermedia applications to provide hypermedia services, with the focus of the paper being on the third method. Section 2 presents a brief review of hypermedia data models to provide an understanding of the objects that must be supported. Section 3 provides an overview of two methods of retrofitting an existing application. These methods are based on traditional hypermedia architectures. Section 4 presents a new mechanism for retrofitting existing applications that results in an open and integrated application environment. A model of the mechanism, implementation details, and procedure for converting applications to utilize the mechanism are discussed. Section 5 presents some of the concerns and constraints that guided the research, limitations of the mechanism, and future research. Section 6 summarizes the paper.

2 MODELS AND IMPLEMENTATIONS

Two general philosophies, prescribed in the form of *data* and *process* models, dominate the field of hypermedia. One philosophy is based on a static or passive model and is typical of *first-generation* hypermedia systems (see [7] and [8] for an overview of hypermedia and a discussion of first-generation behavior). The other philosophy is based on an active or process model. An active model incorporates all of the characteristics of a passive model and extends the static model by adding *computation*.

Computation allows the hypermedia system to dynamically determine the content of nodes and objects, and to determine 'on the fly,' the existence of object relationships [9–12]. A computational model of hypermedia provides significantly more power to the application.[1] However, to fully benefit from the capabilities of a computational model, applications must be designed and constructed with this model in mind so that every component of the application can take full advantage of its power. Realistically, not all applications need computational capabilities. For example, if an application only needs the ability to create and navigate simple links, computational services would be unnecessary.

First-generation hypermedia data models

We categorize the data models of first-generation hypermedia systems into two groups — *attribute-value* and *anchor-link*. An attribute-value model of hypermedia uses attributes and values to maintain relationships among objects. For example, an attribute such as *FontFamily* having a value of *Helvetica* would provide a font definition for a text object. If this object also has a *LinkDest* attribute having a value of *ND43*, the object could represent a link endpoint to destination node ND43. When an object can have an attribute/value pair that holds the link destination to another document or object, we say that the data model for the object 'fits' the attribute-value hypermedia data model.

Adding an attribute/value to an object, however, is not sufficient for classifying an application as 'hypermedia.' Functionality (semantics) must be added to maintain the link attribute and traverse the link (see [13] for a discussion of link semantics). From an interface perspective, the user must be able to identify (by selection) the destination of the link so that when they choose the object in conjunction with navigation (usually by 'mousing' on it), the application follows the link to the destination resulting in the display of the destination objects. If a value is absent from the attribute, no link exists and hence mousing on the object does not result in navigation.

Variations on the attribute-value model enhance the navigational aspects of a first-generation system. For example, allowing multiple values for the attribute would provide the user with a choice of link destinations. When an object having multiple destinations is moused, a menu can be displayed to allow the user to select the destination appropriate to the task. The Intermedia system offers this functionality [14, 15]. As another example, HyperTIES [16, 17] uses a two-stage link traversal method. The user begins by selecting the link to follow. This causes the system to display a short description, provided by the author, which describes the contents of the destination node. At this point the user can

[1] The term 'application' refers to the component of a system that controls activity and allows a user to view application objects. Other major components of a system include the *interface* and *back-end (storage manager).*

either follow the link to the destination, or break out of the navigation and resume normal viewing.

The second most common model of hypermedia is the anchor-link model. This model is based on *anchors* and *links* [7, 18]. An *anchor* associates an application object with a link while a *link* associates two or more anchors. The anchor object contains the identifiers of the application and link objects, thus relating the entities. A link object contains anchor identifiers. The anchor-link model can be extended to allow multi-valued relationships among objects, anchors, and links. This allows the association of an application object with multiple anchors, and hence, multiple links, destinations, and destination objects.

3 PREVIOUS WORK

The data models form the basis of the data management characteristics of a hypermedia system. To retrofit or migrate an existing application to a hypermedia domain, the application is modified to support a data model and carry out the activities of hypermedia such as link selection and navigation.

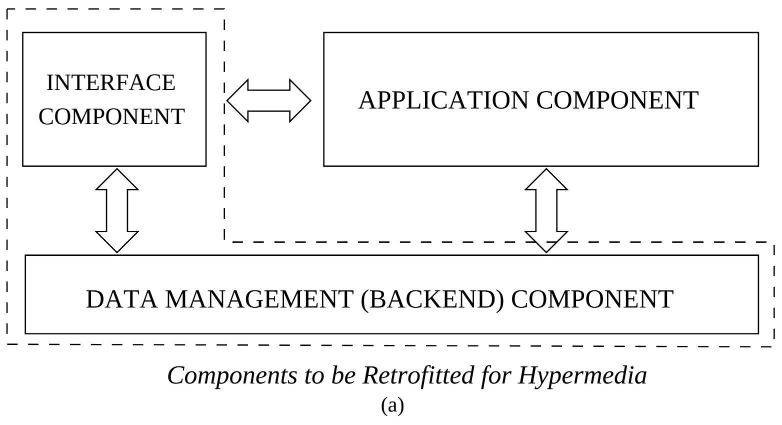
How an application is converted determines the depth of hypermedia activity that the application can provide the user. In this section, we provide a brief overview of two conversion methods that are based on traditional hypermedia system architectures. Both methods involve modifying, building, or rebuilding some of the major components of an application and for this reason are ‘costly’ in the sense that significant effort is needed to realize either method.

The first strategy (see Figure 1(a)), developers must convert the interface and/or backend components to embed hypermedia functionality *within* these components. In the second strategy (see Figure 1(b)), developers must modify the application to interact with a process, external to the application, which handles the management of hypermedia objects. Both strategies are highly effective, especially since applications can supply a ‘rich’ set of messages concerning their activity. This allows both strategies to support second-generation hypermedia services. (See [10] for a discussion of second-generation services.)

Referring to Figure 1(a), this retrofit strategy depends upon the the interface and application components operating independently. Hypermedia functionality is embedded within two supporting components of the application — the interface and a ‘backend’ or storage manager component that is accessible to the interface component.

Retrofitting an application according to this architecture requires several conditions. The interface must distinguish objects with links from those without and display link markers appropriately. Bieber employs this strategy effectively in the MAX system by analyzing display messages that pass between the interface and application components, recognizing and acting on messages that involve objects with links [9, 19, 20]. The advantage of this approach is that the interface can assume total responsibility for displaying link markers, or it can interact with the application to allow the application to display the link markers. The former approach minimizes interface/application interaction while the latter approach enables the application to control its display [21].

Application objects must have a unique identity or attributes that allow them to be distinguished in some way. This is a common property of object-oriented or object-based environments [22]. Whenever the application displays an object, the interface is provided


INTERFACE COMPONENT
APPLICATION COMPONENT
DATA MANAGEMENT (BACKEND) COMPONENT
Components to be Retrofitted for Hypermedia
(a)

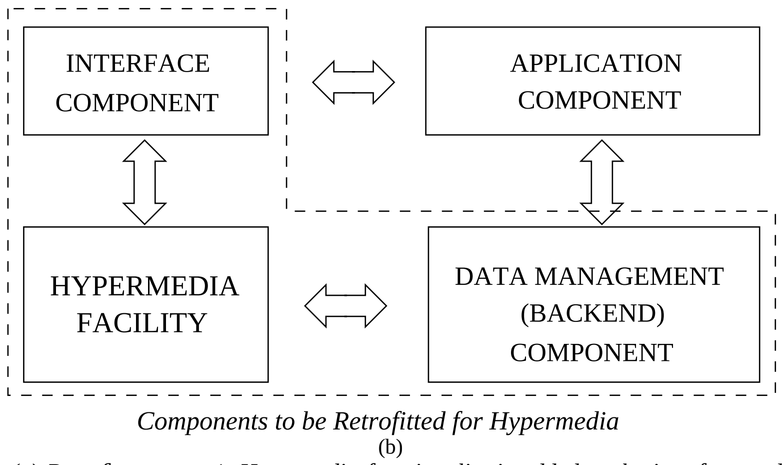

INTERFACE COMPONENT
APPLICATION COMPONENT
HYPERMEDIA FACILITY
DATA MANAGEMENT (BACKEND) COMPONENT
Components to be Retrofitted for Hypermedia
(b)

Figure 1. (a) Retrofit strategy 1. Hypermedia functionality is added to the interface and backend components of an application; (b) Retrofit strategy 2. An external hypermedia engine provides hypermedia services to the user through the interface component of the application

with the object's identifier and uses this information to determine whether the object has links. If a link exists, the object is displayed with or as a link marker.

The application must perform navigation by acting on a directive from the interface. That is, when the user causes a navigational event to occur, the interface resolves the destination (the application is unaware of this activity), producing a list of one or more objects (e.g., document, file, object, etc.) to display. This information is passed to the application upon which the application retrieves and displays the objects.

The application must accept a directive that contains the identifiers of the objects that are to be displayed when a link is followed. There are several ways this can be supported. A communication channel could be established between the interface and application to transmit the request [23]. This implementation method is necessary if the application and interface are physically independent components. Another approach is to handle a navigational event as a normal interface event, like a button or key press. Bier [3] demonstrates and discusses this type of architecture and interface/application interaction in the context of a button interface. Using this approach, a navigational event would cause the execution of a procedure within the body of the application. Parameters to the procedure would identify the object on which the navigational event occurred.

The interface and application components must utilize the services of a data manager. Although toolkit approaches [1, 24] can provide the necessary hypermedia behavior, the application component becomes involved by calling on hypermedia functionality within the toolkit. Thus, the interface is subordinate to the application, that is, the application's interface contains or interacts with a data manager that supports hypermedia objects. This data manager can be different from the data manager for application objects, and in fact, the application may have no knowledge of a hypermedia model or hypermedia services at all. If two different data managers are used, integrity may be jeopardized whenever the application performs an operation but does not inform the interface of the event. For example, if the application deletes an object that is currently displayed and does not inform the interface of the deletion, the link for the object will still exist. In fact, the user can see the object and may be able to navigate links even though the object no longer exists in the application's data store. Bieber solves this problem by interrogating all messages that pass between the interface and application components [9, 19, 20].

The interface must be cognizant of application object structure and content so that link markers are placed appropriately. Link markers must be placed in a reasonable proximity to the object or the object itself must be displayed differently. This can be difficult, especially for complex or composite objects whose content may be based upon other objects. Simple (flat) objects with regular shapes will present the least difficulty in link marker placement for the interface.

Navigation is supported by the single directive *from* the interface *to* the application. The communication channel is often one-way and does not allow the application to benefit from other information in a hypertext. This limits the application (and user) from obtaining information about object relationships, strength of relationships, or from determining an ordering of nodes from a given starting point by following a depth- or breadth-first link traversal algorithm. Although these services could be added, they require the application to have processing capabilities that are beyond a simple navigational model. Adding these capabilities would essentially require converting the application into a full-fledged hypermedia system.

Further, the primary disadvantage of this strategy is that an open and integrated application environment cannot be achieved. Since the application's interface operates as an independent entity, modifications for one application's interface do not affect other applications unless the other applications also utilize that interface. In fact, even if all application interfaces were modified to support this model, cross-application linking still would be difficult due to the lack of an overall umbrella for inter-application linking.

The second retrofit strategy (see Figure 1(b)) is similar to the first except that the hypermedia engine is external to the application and interface components. The interface communicates with the hypermedia engine through services of the operating system or windowing environment. Extensive work has occurred in developing hypermedia systems that could be used to retrofit existing applications according to this architecture. Some of these efforts include the Sun Link Service [25], PROXHY [21], the HBx and SPx systems [13, 26, ABC/DGS [27, 28], HyperForm [29], SEPIA [30, 31], Microcosm [5, 32, 33], and Chimera [6].

To use any of these systems in a retrofitting situation, the application component is modified or augmented to support some level of interaction with the hypermedia engine.

For example, Microcosm and ABC require that a custom menu be attached to an application's window. In the case of Microcosm, the menu is attached through macro services provided by or added to the underlying application. ABC uses the window hierarchy services of the X Window System [34] to automatically add a menu to the application's window, without any awareness of the menu by the underlying application. Microcosm and ABC both depend upon *selection* or *clipboard* services of the windowing environment to identify objects to which links should be attached.

Chimera, PROXHY, and the SPx suite of tools utilize services of the operating system to support interaction between the application component and hypermedia engine. To retrofit an existing application to use these systems, the application is enhanced to interact with the hypermedia server and respond to server requests. Chimera depends upon remote procedure call (RPC), PROXHY uses interprocess communication services (sockets), and SPx uses interprocess communication services (sockets) contained within the windowing environment.

In all cases, these approaches can minimize the interactions between the application component and hypermedia engine thus reducing the extent of modification necessary to retrofit an existing application with hypermedia services provided by their engines. This also allows these facilities to support applications that cannot or do not need to be extensively modified to provide some level of hypermedia activity. To illustrate, ABC can distribute and share an application window with other users without any intervention, participation, or knowledge of this capability by the underlying application. This is accomplished by intercepting and distributing window system interface events to other displays.

As with the retrofit strategy shown in Figure 1(a), the behavior among the application, interface, and hypermedia components is similar. The application component typically accepts directives that cause a link marker to appear alongside an object or the object itself to appear differently, and to identify the objects the application should display when a link is followed. These directives are sent *from* the hypermedia engine *to* the application component, possibly *through* the interface component.

4 PROCESS-BASED RETROFIT STRATEGY

The primary goal of this research was to investigate the ability to model and provide hypermedia services to existing applications while minimizing the effort needed to modify the application to support a navigational paradigm. Achieving this goal would allow developers and users to experiment with hypermedia services without incurring expensive application conversion costs. The term *cost* refers to the difficulty and extent of modification that must be made to an application.

As discussed, the previous retrofit strategies depend upon one of two general conditions: (1) the interface, application, and/or backend components are enhanced to display link markers, interrogate hypermedia-related messages that pass among the components, and act on navigational events, or (2) the interface and/or application components are augmented through a set of customized macros or menus, anticipated by developers and built into the application or within the operating environment. But, what if extensive modification to the application or its interface cannot occur, such as in the case of a commercial system? For example, reconsider the application that displays scanned images of book pages or maps discussed in the *Introduction*. How can this application be enhanced to support hypermedia?

We now present a third retrofit method, based on a process model of hypermedia and on the premise that a very ‘sparse’ set of messages pass between the application and hypermedia components. This retrofit strategy has an architecture that is similar to strategy #2 (Figure 1(b)) but differs in two major ways (1) it can be accomplished with **no** modification or a trivial modification of the application component, and (2), it requires **no** modification of the application’s backend or data store.[2] This approach results in an application that is completely unaware of interface and data management activities associated with hypermedia, but behaves like a first-generation hypermedia system.

This retrofit strategy is based on an integration of the attribute-value and anchor-link models, that is, the data model is based on the attribute-value model while the implementation is based on a process-oriented, anchor-link model. Characteristics of this approach include: (1) links are maintained as attribute/value pairs; (2) navigation requests are ‘trapped’ by a process that manages the display of link markers; (3) link destinations are resolved by the hypermedia engine; (4) the appearance and presence of a link marker is determined and supported by a process that is external to all components of the application; (5) data associated with anchors and links are managed by the hypermedia engine and its backend; (6) the link destination is delivered to the application through the windowing environment or through a file. In essence, the windowing system and window manager comprise the destination anchor process in this process model architecture, delivering the destination of the link to the application.

Other multi-application and process-based approaches are similar (also see earlier discussion under retrofit strategy 2) in that they externalize service (e.g., hypermedia engine, event handler) and the application components. For example, nodes in Multicard [35] are under the control of editors (applications) and applications must support the M2000 protocol in order to interact with the hypermedia component. This allows Multicard to support links *within* nodes as well as links between nodes. In contrast, no application components actively participate in supporting hypermedia in this retrofit strategy. In fact, the overall goal of this strategy is different in that the primary objective is to provide a framework to allow developers to *experiment* with hypermedia as a possible access paradigm for existing applications — not to provide a complete hypermedia environment.

Grif [36] and Hyperform [29] use event or notification services to inform users or components of an application or activity within the operating environment. In Grif, events can effect document management at multiple levels, for example, causing the transformation of a document to another form or the deletion of a paragraph of text. Hyperform uses notification to support collaboration, allowing a user to register and receive specific events about specific objects. In each of these approaches, events pass *in to* and *out of* component boundaries. This enables the application to support highly structured and complex processing, but also depends on the application component to handle and generate events to continue work. Our assumption was that the application did not and could not possess this level of event handling. Moreover, we assumed that the application could not be modified, without significant effort, to achieve this level of interaction. Hence, the *only* event delivered from the facility to the application is to display a node.

The key to the architecture and operation of this retrofit strategy is the encapsulation of hypermedia functionality into two components — the windowing environment and an

[2] The software is available via anonymous ftp from ftp.cs.fsu.edu:/pub/hypertext/freckles.tar.Z. The software is Sun Sparc 4 compatible and executes under the *twm* X Window System manager.

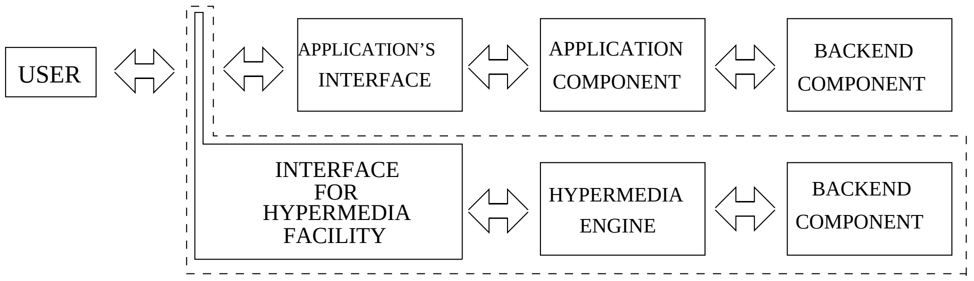


Figure 2. Retrofit strategy 3. Hypermedia services are supported by a separate entity (enclosed in dashed region) which resides completely outside the application and which 'taps' into the user↔application communication channel. This strategy depends on specific services of the window manager and windowing environment for support

external hypermedia facility. We use the term *facility* since this element interacts with the window manager, contains or interacts with a hypermedia object manager or engine, and uses services of the windowing environment or operating system to send a message to the application component. No messages pass *from* application components *to* the facility.

The hypermedia facility is implemented as a separate, autonomous process and executes simultaneously with application components and the window manager. The hypermedia facility is completely autonomous and interacts directly with the user (see Figure 2). Although the autonomy and intent of the hypermedia facility are similar to the *omniscient monitor* [37], this retrofit strategy differs significantly in that the hypermedia facility has no knowledge of activity within an application and does not understand the application's data model.

The hypermedia facility monitors the activities in windows on the user's workstation, similar to the behavior of the facility that supports collaboration in the ABC system [28]. Either specific windows can be identified or the facility can be instructed to monitor all windows. Whenever a window is opened, closed, or moved, the hypermedia facility is notified of the event by the window manager. Thus, important aspects of this retrofit strategy are the windowing environment and window manager. The facility must interact with the window manager in order to obtain information about the existence of windows, specifically the title of a window, and the location of each window on the display. Window managers that cannot provide these services cannot be used with this facility.

The window title serves as the identifier for the window's contents, allowing the hypermedia facility to create and maintain links based on the value of the title string. Applications can affect the granularity of links by changing the window title to reflect different portions of nodes that are displayed. The title is used to determine if links exist for the window by accessing a hypermedia data store. If links exist, the facility retrieves the link information and displays link markers 'on top of' the application's window, attaching the links to the window as subwindows. The link markers appear as an integral part of the application's window and remain attached to the window even if the window is moved by the user, application, or window manager. The position of a link marker is determined by the user when it is created.

A separate menu, provided by the hypermedia facility and separate from the application menus, allows the user to control the facility and manage links. The menu appears as a typical menu on the display. Since the appearance and placement of the menu can be tailored to coordinate with application displays, it is extremely easy to configure the menu and execute the facility so that the user is not aware that two separate processes are working together to support hypermedia services.

The hypermedia facility must be supported by a data manager. The objects necessary to support the facility are attribute/value-based entities having a unique identifier. Any of the previously mentioned external hypermedia engines/servers (HBx, SPx, SEPIA, Microcosm, Multicard, and Ham [38] etc.) would be more than sufficient to support the data management needs of the facility.

Prototype

A prototype of retrofit strategy #3 was implemented in the X Window System [34] environment on Sun[3] workstations using the *twm* window manager. The facility was implemented in C and has been tested with applications written in C and C++. The applications provide text editing and image display services.

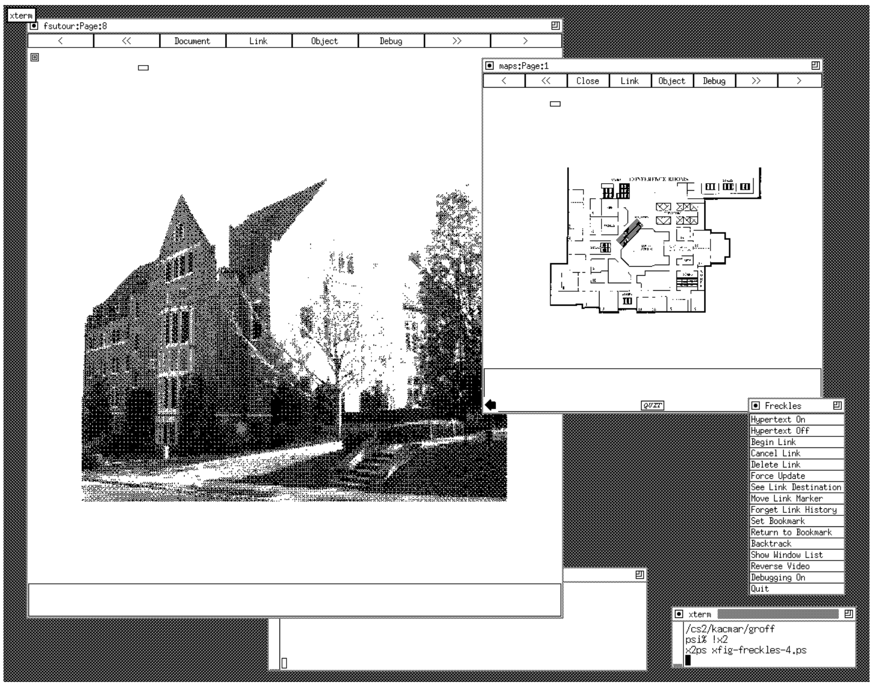
Upon startup, the facility obtains the name of the process driving each window and the window's title by interrogating internal data structures within the windowing environment. A resource file identifies the application windows that will be automatically and continuously monitored. If an application's window is listed in the resource file, the facility uses the value of the title as an object identifier (key) into the hypermedia data store. If the window title identifies an 'object' in the store, links exist for that window; the links are retrieved, and instantiated as subwindows on top of the existing application's window. Link markers appear either as small unfilled rectangles (see Figure 3), as rectangles containing the destination address/title (see Figure 4). Link markers are colored to distinguish them from other information on the display. For all applications, link markers appear in the same style and color, and all link activities are consistent. This results in a consistent presentation of the hypermedia model and interface activities to users and applications.

Menu-based services

The menu for the hypermedia facility appears in the lower right of the display in Figures 3 and 4 (it can be positioned anywhere). It is used to support the following services:

- turn hypermedia facility 'on' or 'off';
- define the source and destination ends of a link;
- cancel a link — ignore link definition request;
- force an update on all link markers;
- change the appearance of link markers;
- move a link marker;
- forget the backtrack history;
- set and return to a bookmark;
- backtrack to the most recent node visited;
- shutdown the hypermedia facility (quit).

[3] Sun is a trademark of Sun Microsystems, Inc.



```
/cs2/kacmar/groff
psi% !x2
x2ps xfig-freckles-4.ps
```

Figure 3. Link markers appear as hollow rectangles and are visible in the upper left regions of the building and floor plan images. The menu for the hypertext facility is in the lower right

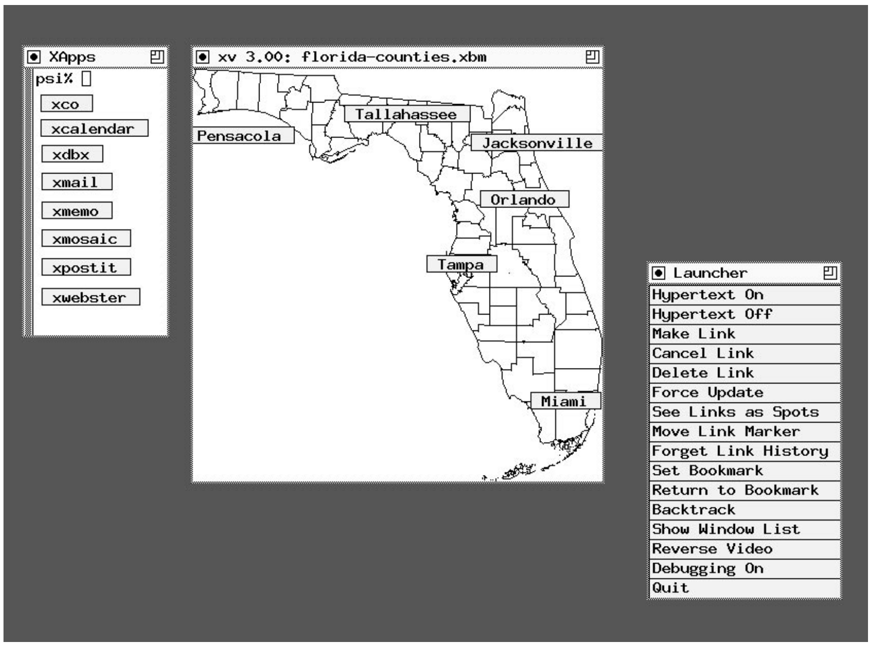


Figure 4. Link markers may be set to display the destination of the link. Shown here is the version of the facility that launches applications. An xterm application (left) is populated with links that invoke various X applications. The facility uses the window title to associate these links with the object — 'XApps'. The xv application (center) is displaying a map of Florida that is populated with links to subordinate maps. The links in this window invoke shell scripts that launch each of the respective city maps. Notice that the window title in the xterm application is a single contiguous string, while the window title format in the xv application consists of multiple segments

Hypermedia 'on' and 'off'. The user can turn the hypermedia facility on and off for selected windows on the display. These menu items can be used in lieu of identifying the application windows that must be monitored through the startup file.

Define link ends. A link is defined by clicking the Begin-Link menu item. The user then moves the cursor into the application window and clicks the mouse button at the position at which the link marker should appear. The facility obtains (from the window manager) the title of the window and location of the mouse click. These elements determine the placement of link markers when this window is displayed in the future. Once the source end of the link has been defined, the user defines the destination linkend using the same procedure. Links are bidirectional and span application boundaries.

Cancel a link. If the user creates a source end of a link but decides to not complete the link (by creating a destination end), this menu item allows the user to cancel the link request. The source end of the link is deleted and the link marker is removed from the window.

Delete a link. The user mouses the menu item and then clicks on the link marker to be deleted. The link associated with the link marker is removed from the hypertext, effectively deleting the link and its anchors. An alternative implementation might delete only one anchor.

Force update of the link markers. Occasionally, a link marker becomes obscured by a subwindow of the application. Clicking this menu item causes the facility to refresh the link marker display. This was not a planned feature of the facility but rather a way to get around a (rare) conflict with the window manager.

Change link marker appearance. Link markers can be displayed in two ways — as unfilled (hollow and colored) rectangles or as rectangles with the target window title (destination) displayed. The user can toggle the format of the link marker display by clicking on this menu item.

Link history, bookmarks, and backtracking. The hypermedia facility maintains a history of nodes visited. The user can use this information to backtrack. The user can instruct the facility to forget the history causing the current location to become the 'root' of navigation. Bookmarks can be dropped anywhere along the path. Clicking on the Return-To-Bookmark menu item causes the user to return to the most recently set bookmark — the bookmark is also removed. Of course, the operation of this bookmark facility may differ from others.

Application modification

Only one requirement must be honored for an application to comply with this facility — the application must accept a directive that identifies the objects to display after a link is followed. Two methods can be used to transmit this directive into the application. No messages flow *from* the application *to* the facility.

Method 1 — common file. The application monitors the file system for the existence of a specially-named file. When a link is followed, the hypermedia facility creates the file and writes the identities of the destination objects into it. The application reads the file, 'navigates' to the destination by displaying the objects, and then deletes the file. Mosaic uses a similar approach in its *common gateway interface* [39]. Using this method, **no** modification of the application is needed to participate in the retrofit strategy.

Method 2 — window property event. Three simple modifications must be made to an application to use this method of interaction with the facility. As shown in Figure 5, the modifications involve: (1) definition of an X window system data element; (2) initialization; and (3) event handling. Modification (3), event handling, is the most important since it is the modification that allows the application to carry out navigation. The reader can consult Chapter 12 of Jones [40] for details on the X elements and directives.

Data definition
```
Atom     htnavigateatom;
```

Initialization
```
htnavigateatom =
    XInternAtom(display, "HT_NAVIGATE", False);
```

Window event handle
```
case PropertyNotify:
    if (event.xproperty.atom == htnavigateatom){
       XGetWindowProperty(display, window,
           htnavigateatom, 0, 8192, True, XA_STRING,
           &returned_atom, &returned_format,
           &returned_itemnos, &bytes_return,
           &returned_value);
       if (returned_value != NULL){
           destination is in returned_value
       }
    }
```

Figure 5. The modifications that must be made to an application to comply with the retrofit strategy

In a traditional hypermedia system, a navigation event requires that the destination of the link be resolved and the application component display the objects at the link destination. These activities are realized in retrofit strategy #3 through interactions between the hypermedia facility and window system (the application and its components play a minimal role). The hypermedia facility recognizes a navigational request when the user clicks a mouse button in a link marker window, resolves the link destination, and notifies the application process of a navigational request *through an X window system property event*. The property event returns to the application the identifier of the link destination as a character string. This value must be interpreted by the application, resulting in the display of the objects that constitute the destination. Since all X applications must have the functionality to handle events and display objects anyway, compliance with this retrofit strategy requires only the simple modifications described above. In fact, any X-compliant application can participate in hypermedia services since the behavior of the facility is the same for all applications. This contributes to the open system architecture and an integrated application environment.

5 ISSUES, DISCUSSION, AND FUTURE RESEARCH

Several secondary goals were investigated during the research. These include: to determine the limitations of the process-based retrofit strategy; to assess the ability to coordinate and maintain the consistency of displays for multiple applications, and to identify extensions of the facility.

Seamless, open, integrated architecture

The process architecture and operation of the facility achieves the goals of a seamless, open, and integrated application environment in several ways. First, since the facility is responsible for the management and display of link markers, all application windows are populated with link markers in exactly the same way. Second, users define, activate, and perform other link operations by interacting with the hypermedia facility through the menu or link markers. Hence, the facility provides link services in a consistent manner for all participating applications. Third, participation is not dependent on application architecture or interface style. Of course, the application must meet the minimal conditions for operation — by accepting the navigate directive from the facility following link navigation. Fourth, links can be defined across application boundaries. Since the hypermedia facility is external to all applications, link operations do not depend on application data or activities.

Performance

One of our major concerns was performance. Since the hypermedia facility executes in parallel with application processes and can serve more than one application at the same time, we were concerned that either overall application performance would be impacted or there would be significant delays in the removal and/or display of link markers. However, neither problem materialized.

The hypermedia facility is idle during application activity and becomes ‘active’ only when *certain* interface events occur on an application’s window or when activity occurs on the menu of the facility. Minimal resources are used during the sleeping period resulting in no degradation of application performance. The amount of work required to remove link markers, evaluate the window title, and display new link markers is minimal. In fact, link markers sometimes appear faster than the application can display its objects. This is especially true and noticeable when the application is displaying a large image. There is, however, a slight delay in removing link markers from a window following a change in window contents.

Link identification

The window title plays a key role in this strategy, serving as the identifier of the window’s contents. Application activities can conflict if (1) a link is followed from one application to another and the destination specifies an object that is unknown to the target application; or, (2) the title is formatted in a special way that requires decoding and the target application is unable to decode the title. To avoid these problems, it is suggested that all applications adopt a common method of formatting the title and identifying objects. The *xv* application (see Figure 4, center window) is an application whose window title format conforms to this specification.

Link identification through the window title provides an opportunity for future research if applications are responsive to the new format. For example, the applications shown in Figures 3 and 4 use a structured window title format as follows:

application identifier : context : object identifier

This format allows the facility to identify the name of the application, object, and context for this window. When a link is followed, applications can extract a context identifier, which could be used to filter link markers during navigation. That is, if the source end of the link is created in the context 'smith' and the destination is created in the context 'jones,' following a link would cause the application to switch contexts. Thus, in a very simple fashion, applications could use the facility to support views over the information space.

Link granularity

Currently, the granularity of a link endpoint is a node. This is appropriate and sufficient for the example applications (map and page-image display) discussed throughout this paper, but may not be sufficient for other applications. As discussed, the node-to-node linking model is adequate for applications that use a data model where the granularity of a node is equal to the size of a window. This approach is also effective for nodes larger than a window, provided the application uses the window title to identify the object and the current viewport within the node. Since most applications use the window title to identify the current object being displayed, adding viewport information would require minimal additional work.

The current prototype cannot support link markers for objects that are embedded 'within' a window because the presence or identities of objects within the window cannot be determined and monitored using existing window managers. This also means that it is not possible to use this facility in an application where the position of objects change, such as an application that supports scrolling. These disadvantages may be solved in future research. There are at least two approaches to this investigation. First, enhancing the facility to 'recognize' objects in a node, using an object's features to place link markers in close proximity. This approach would be based on work in object recognition and image analysis for hypermedia environments [41]. Second, enhancing the window manager and windowing system to allow applications to identify objects when they are written to the display. This approach is based on previous work of the author in which objects are identified at the interface level [4]. As long as each object has a unique identity and the window manager maintains a list of displayed objects with their positions, the facility can interact with the window manager to determine the object to which a link end should be associated.

Link semantics

The prototype currently supports bidirectional links. We realized during our research that it would be possible to support other types of links. By allowing the user to specify the link type when a link is created, links that launch applications, initiate events, or perform other process-oriented activities could be defined. However, since links and link activity are external to an application, the communication channel between the facility and the application must be widened to support link types that effect activities that are *internal* to an application. Such a modification is contrary to the intent of the facility.

We prototyped and experimented with a second version of the facility (see Figure 4) where links are used to launch applications. Extending the facility to support this behavior was simple. We also realized that the process-oriented model of the facility could support computed destinations and dynamic creation of nodes (see [10] and [18] for an explanation of these activities). Although the process approach would support parallel or serial object display and composition, as in AHM [42], previous work has reported difficulties in coordinating parallel activities across processes [21].

6 CONCLUSION

Existing applications can benefit from hypermedia services but the cost of converting these applications to hypermedia can be expensive. Conversion normally requires modification of the interface, application, backend components, and enhancement of the application's data store to maintain the objects that form associations. The goal of this research was to provide developers with a model and method of experimenting with hypermedia services, while minimizing the cost of conversion. This would allow developers to assess the impact of hypermedia services on users, and if appropriate, plan for full conversion of the application to support complete hypermedia activities.

This paper discussed three methods in which an application can be converted/retrofitted to provide hypermedia services. Two of the methods are based on traditional hypermedia architectures. We contend that retrofitting an application using either of these approaches would be 'costly' in the sense that the application, the application's interface, or the storage manager must undergo modification to interact with a hypermedia engine and to exchange and display information about objects and link markers.

The third method, and focus of the paper, significantly reduces the cost of conversion and is successful in achieving the goals of the research. This approach retrofits an application without modifying the application's interface or its storage manager. In fact, depending on how the application is constructed, no modification may be necessary. The method results in an open and integrated hypermedia environment in which links connect applications and nodes within applications. The hypermedia engine and link marker interface functionality are located completely outside the application, in a separate and autonomous hypermedia process, providing consistent link marker appearance and link behavior for all applications served by the facility.

ACKNOWLEDGEMENTS

The author would like to thank John Whitley for the initial implementation of the prototype, and John Schnase, Michael Bieber, Antoine Rizk, and the anonymous reviewers of *EP-odd* for their excellent suggestions on previous drafts.

REFERENCES

1. J. Puttress and N. Guimaraes, 'The toolkit approach to hypermedia', in *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext*, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, INRIA, France, November 1990, pp. 25–37.
2. G. Cockton, 'A new model for separable interactive systems', *Proceedings of the Second IFIP Conference on Human-Computer Interaction — INTERACT '87*, Stuttgart, FRG (1986).
3. E. Bier, 'EmbeddedButtons: documents as user interfaces', *Proceedings of the ACM*

Symposium on User Interface Software and Technology – UIST '91, Hilton Head, SC, pp. 45–53 (1991).

4. C. Kacmar, 'Supporting hypermedia services in the user interface', *Hypermedia*, **5**(2), 85–101 (1993).
5. H. Davis, S. Knight, and W. Hall, 'Light hypermedia link services: a study of third party application integration', *Proceedings of the 1994 European Conference on Hypermedia (ECHT)*, Edinburgh, Scotland pp. 41–50 (1994).
6. K. Anderson, R. Taylor, and E. Whitehead, 'Chimera: hypertext for heterogenous software environments', *Proceedings of the 1994 European Conference on Hypermedia (ECHT)*, Edinburgh, Scotland pp. 94–107 (1994).
7. J. Conklin, 'Hypertext: an introduction and survey', *IEEE Computer*, **20**(9), 17–41 (1987).
8. N. Meyrowitz, 'The missing link: Why we're all doing hypertext wrong', in *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information*, ed. E. Barrett, The MIT Press, Cambridge, MA, 1989, pp. 107–114.
9. M. Bieber, hypertext interface"" 'Issues in modeling a "dynamic" hypertext interface', *Proceedings of the Hypertext '91 Conference*, San Antonio, TX, pp. 203–218 (1991).
10. F. Halasz, 'Reflections on NoteCards: Seven issues for the next generation of hypermedia systems', *Commun. ACM*, **31**(7), 836–852 (1988).
11. J. Schnase and J. Leggett, 'Computational hypertext in biological modeling', *Proceedings of the Hypertext '89 Conference*, Pittsburgh, PA, pp. 181–197 (1989).
12. P. Stotts and R. Furuta, 'Dynamic adaptation of hypertext structure', *Proceedings of the Hypertext '91 Conference*, San Antonio, TX, pp. 219–232 (1991).
13. J. Leggett and J. Schnase, 'Viewing Dexter with open eyes', *Commun. ACM*, **31**(2), 81 (1994).
14. N. Meyrowitz, 'Intermedia: the architecture and construction of an object-oriented hypermedia system and applications framework', *Proceedings of the OOPSLA '86 Conference*, Portland, OR, pp. 186–201 (1986).
15. N. Yankelovich, B. Haan, N. Meyrowitz, and S. Drucker, 'Intermedia: the concept and the construction of a seamless information environment', *IEEE Computer*, **21**(1), 81–96 (1988).
16. B. Shneiderman, 'User interface design for the HyperTIES electronic encyclopedia', *Proceedings of the Hypertext '87 Conference*, Chapel Hill, NC, pp. 189–194 (1987).
17. B. Shneiderman, C. Plaisant, R. Botafogo, D. Hopkins, and W. Weiland, 'Designing to facilitate browsing: a look back at the Hyperties workstation browser', *Hypermedia*, **3**(2), 101–117 (1991).
18. F. Halasz and M. Schwartz, 'The Dexter hypertext reference model', *Commun. ACM*, **37**(2), 30–39 (1994).
19. M. Bieber, 'Automating hypermedia for decision support', *Hypermedia*, **4**(2), 83–110 (1992).
20. M. Bieber, 'On integrating hypermedia into decision support and other information systems', *Decision Support Systems* 251–267 (1995).
21. C. Kacmar and J. Leggett, 'PROXHY: a process-oriented extensible hypertext architecture', *ACM Trans. on Inf. Syst.* (4), 399–419 (1991).
22. S. Khoshafian and G. Copeland, 'Object identity', *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications — OOPSLA '86, Sigplan Notices* (11), 406–416 (1986).
23. C. Fraser, 'A generalized text editor', *Commun. ACM* (3), 154–158 (1980).
24. M. Sherman, W. Hansen, M. McInerny, and T. Neuendorffer, '', in *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext*, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, INRIA, France, November 1990, pp. 13–24.
25. A. Pearl, *Proceedings of the Hypertext '89 Conference*, Pittsburgh, PA, pp. 137–146 (1989).
26. J. Schnase, J. Leggett, and D. Hicks, '', Department of Computer Science Technical Report No. TAMU 91-003, Texas A&M University, College Station, TX (1991).
27. D. Shackelford, J. Smith, and F. Smith, *Hypertext '93 Conference Proceedings*, Seattle, WA, pp. 1–13 (1993).
28. K. Jeffay, J. Lin, J. Menges, F. Smith, and J. Smith, *CSCW '92 Conference Proceedings*, Toronto, Canada, pp. 195–202 (1992).
29. U. Wiil and J. Leggett, *European Conference on Hypertext (ECHT) '92 Proceedings*, Milan, Italy 251–261 (1992).
30. H. Schutt and N. Streitz, '', in *Hypertext: Concepts, Systems and Applications, Proceedings of*

the European Conference on Hypertext, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, INRIA, France, November 1990, pp. 95–108.

31. N. Streitz, J. Haake, J. Hannemann, A. Lemke, W. Schuler, H. Schutt, and M. Thuring, *Proceedings of the European Conference on Hypertext (ECHT '92)*, Milan, Italy, pp. 11–22 (1992).
32. A. Fountain, W. Hall, I. Heath, and H. Davis, '', in *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext*, ed. A. Rizk, N. Streitz, and J. Andre, Cambridge University Press, INRIA, France, November 1990, pp. 298–311.
33. H. Davis, W. Hall, I. Heath, G. Hill, and R. Wilkins, *European Conference on Hypermedia (ECHT) '92 Proceedings*, Milan, Italy, pp. 181–190 (1992).
34. R. Scheifler, J. Gettys, and R. Newman, , Digital Press, Bedford, MA, 1988.
35. A. Rizk and L. Sauter, *Proceedings of the European Conference on Hypertext (ECHT) '92*, Milan, Italy, pp. 4–10 (1992).
36. V. Quint and I. Vatton, 'Making structured documents active', *Electronic Publishing*, **7**(2), 55–74 (1994).
37. P. Balasubramanian, T. Isakowitz, H. Johar, and E. Stohr, 'Hyper model management systems', *Proceedings of the 25th Hawaii International Conference on System Sciences (HICSS)*, Kauai, HI, pp. 462–472 (1992).
38. B. Campbell and J. Goodman, 'HAM: a general-purpose hypertext abstract machine', *Commun. ACM*, **31**(7), 856–861 (1988).
39. NCSA/Mosaic, *Mosaic User's Guide*, Available from University of Illinois, Champaign, IL., 1994.
40. O. Jones, *Introduction to the X Window System*, Prentice Hall, Englewood Cliffs, NJ, 1989.
41. K. Hirata, Y. Hara, N. Shibata, and F. Hirabayashi, 'Media-based navigation for hypermedia systems', *Hypertext '93 Proceedings*, Seattle, WA, pp. 159–173 (1993).
42. L. Hardman, D. Bulterman, and G. vanRossum, 'The Amsterdam Hypermedia Model: adding time and context to the Dexter model', *Commun. ACM*, **37**(2), 50–62 (1994).