HIP: A HYPERMEDIA EXTENSION OF THE PICASSO APPLICATION FRAMEWORK

by

Beverly S. Becker and Lawrence A. Rowe

Memorandum No. UCB/ERL M90/121

19 December 1990

and hypermedia systems.

Several noteworthy features of HIP allow it to achieve these goals and distinguish it from other existing hypermedia systems. Principal among these features is HIP's support for video data. While there are other systems that incorporate video, few provide the level of functionality achieved in HIP. Users can not only follow links into and out of video nodes, but can also use the integrated PICASSO video browser to view any part of the video data at any speed or direction. Moreover, HIP's multidimensional representation scheme allows links to be associated with regions within frames as well as with temporal sequences of frames.

HIP's internal representation scheme is related to another of the system's distinguishing characteristics: extensibility. Many systems claim extensibility, which may mean simply that the system supports the development of new applications, or that the system supports the addition of new media but requires custom-built browsers to do so. HIP provides a more general extension mechanis by defining an abstraction layer that allows new media to be incorporated quickly and easily. Any browser or editor developed in PICASSO can be integrated into HIP.

Finally, HIP provides comprehensive user interface support for creating and browsing hypermedia documents. While many of the features of the HIP interface are found in other systems, the completeness of the set and the sophistication of several of the individual tools are noteworthy. For example, many systems provide a graphical overview map of a hyperdocument; in HIP, this graph is generated dynamically from the data structures representing the hyperdocument, and thus can be updated automatically whenever the document is modified. HIP also provides the path, bookmark, and history list mechanisms common to most hypermedia systems; in addition, it provides a flexible filtering tool that allows the user to view only the nodes and links satisfying specified criteria. The use of filters can be particularly helpful in complex hyperdocuments with hundreds or even thousands of nodes.

The remainder of the paper is organized as follows. Section 2 contains a brief discussion of related work in hypertext and hypermedia. Section 3 introduces HIP through examples that illustrate its noteworthy features. The HIP data model and storage mechanisms are presented in Section 4. Section 5 describes the HIP user interface, focusing in particular on the facilities implemented to reduce user disorientation and aid navigation. The hypermedia abstractions used to integrate HIP with PICASSO are outlined in Section 6. Section 7 discusses the lessons learned in developing HIP and the advantages and disadvantages of using PICASSO as a base for the system. Finally, Section 8 presents our conclusions and discusses plans for future work.

2. Related Work

Dozens of hypertext and hypermedia systems exist today, ranging from relatively simple hypertext browsers to sophisticated multimedia authoring tools. This range of functionality is well-documented by Conklin [Con87]. The former class includes systems such as the Symbolics Document Examiner, a read-only Help system for the Symbolics Lisp environment that includes extensive search and bookmarking mechanisms [Wal85]. The latter class contains systems like Intermedia, a complex hypermedia applications development environment that provides tools for building and navigating multimedia documents [Mey86]. HIP also belongs at this end of the spectrum, providing a full complement of authoring and browsing tools for multimedia hyperdocuments.

HIP: A HYPERMEDIA EXTENSION OF THE PICASSO APPLICATION FRAMEWORK

by

Beverly S. Becker and Lawrence A. Rowe

Memorandum No. UCB/ERL M90/121

19 December 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

HIP: A HYPERMEDIA EXTENSION OF THE PICASSO APPLICATION FRAMEWORK

by

Beverly S. Becker and Lawrence A. Rowe

Memorandum No. UCB/ERL M90/121

19 December 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

HIP: A Hypermedia Extension of the PICASSO Application Framework[1]

Beverly S. Becker
Lawrence A. Rowe

Computer Science Division-EECS
University of California
Berkeley, CA 94720

Abstract

HIP is an extensible hypermedia system built using the PICASSO graphical user interface development system. HIP incorporates text, images, tables, graphics, video, and audio data, and allows users to define new media types easily. This paper describes the HIP architecture and hypermedia capabilities, focusing in particular on its support for continuous media, its extensibility, and the tools it provides to minimize user disorientation.

1. Introduction

The PICASSO application framework is a Lisp-based, object-oriented graphical user interface development system developed at U.C. Berkeley [Row90]. It provides a variety of editors and browsers for displaying and manipulating text, graphics, tables, video, and other data. HIP[2] is an extensible hypermedia framework that is fully integrated with PICASSO.

Authors can use HIP to organize multimedia information into complex *hyperdocuments*. Each hyperdocument contains a set of *nodes*, each of which presents a collection of data in one or more media (e.g., text, tables, images, video sequences). Relationships among the nodes are expressed via typed, directional *links*. Hyperdocument readers can browse this network of nodes by following links from node to node, by selecting nodes from menus or a graphical map of the document, by creating and accessing bookmarks, and by following predefined paths.

HIP has been designed to satisfy three main goals. The first goal was to produce a truly multimedia system, incorporating not only text, graphics, and other static media, but also continuous media (i.e., video and audio). Moreover, these diverse media were to be integrated within a single framework with consistent internal software and user interfaces. The second goal was to minimize the effort required to incorporate new media: well-defined, simple procedures should be provided to facilitate the addition of new node and link types. The final goal was to provide a rich set of

1. This research was supported by the National Science Foundation (Grant MIP-8715557) and The Semiconductor Research Corporation, Philips/Signetics Corporation, Harris Corporation, Texas Instruments, National Semiconductor, Intel Corporation, Rockwell International, Motorola Inc., and Siemens Corporation with a matching grant from the State of California's MICRO program.

2. Hypermedia In PICASSO

and hypermedia systems.

Several noteworthy features of HIP allow it to achieve these goals and distinguish it from other existing hypermedia systems. Principal among these features is HIP's support for video data. While there are other systems that incorporate video, few provide the level of functionality achieved in HIP. Users can not only follow links into and out of video nodes, but can also use the integrated PICASSO video browser to view any part of the video data at any speed or direction. Moreover, HIP's multidimensional representation scheme allows links to be associated with regions within frames as well as with temporal sequences of frames.

HIP's internal representation scheme is related to another of the system's distinguishing characteristics: extensibility. Many systems claim extensibility, which may mean simply that the system supports the development of new applications, or that the system supports the addition of new media but requires custom-built browsers to do so. HIP provides a more general extension mechanis by defining an abstraction layer that allows new media to be incorporated quickly and easily. Any browser or editor developed in PICASSO can be integrated into HIP.

Finally, HIP provides comprehensive user interface support for creating and browsing hypermedia documents. While many of the features of the HIP interface are found in other systems, the completeness of the set and the sophistication of several of the individual tools are noteworthy. For example, many systems provide a graphical overview map of a hyperdocument; in HIP, this graph is generated dynamically from the data structures representing the hyperdocument, and thus can be updated automatically whenever the document is modified. HIP also provides the path, bookmark, and history list mechanisms common to most hypermedia systems; in addition, it provides a flexible filtering tool that allows the user to view only the nodes and links satisfying specified criteria. The use of filters can be particularly helpful in complex hyperdocuments with hundreds or even thousands of nodes.

The remainder of the paper is organized as follows. Section 2 contains a brief discussion of related work in hypertext and hypermedia. Section 3 introduces HIP through examples that illustrate its noteworthy features. The HIP data model and storage mechanisms are presented in Section 4. Section 5 describes the HIP user interface, focusing in particular on the facilities implemented to reduce user disorientation and aid navigation. The hypermedia abstractions used to integrate HIP with PICASSO are outlined in Section 6. Section 7 discusses the lessons learned in developing HIP and the advantages and disadvantages of using PICASSO as a base for the system. Finally, Section 8 presents our conclusions and discusses plans for future work.

2. Related Work

Dozens of hypertext and hypermedia systems exist today, ranging from relatively simple hypertext browsers to sophisticated multimedia authoring tools. This range of functionality is well-documented by Conklin [Con87]. The former class includes systems such as the Symbolics Document Examiner, a read-only Help system for the Symbolics Lisp environment that includes extensive search and bookmarking mechanisms [Wal85]. The latter class contains systems like Intermedia, a complex hypermedia applications development environment that provides tools for building and navigating multimedia documents [Mey86]. HIP also belongs at this end of the spectrum, providing a full complement of authoring and browsing tools for multimedia hyperdocuments.

HIP's design was influenced by many other hypertext and hypermedia systems, including the above-mentioned Document Examiner and Intermedia. Like the Document Examiner, HIP supports user-defined bookmarks; however, HIP allows the user to mark any region as a bookmark and to annotate these bookmarks with meaningful text, whereas the Document Examiner simply stores the title of the section being marked. Intermedia's influence is apparent in the HIP data and interaction models (e.g., the operations for creating and following links) and in the variety of navigation tools provided. Both systems are extensible to new media and applications; however, Intermedia requires custom-built browsers whereas HIP was designed to be integrable with any PICASSO tool.

Other systems whose features are reflected in HIP include the Virtual Notebook System [Shi89]. NoteCards [Hal87], Hypercard [App87], and Muse [Hod89]. The Virtual Notebook System, developed to support collaborative biomedical research, provides a flexible filtering mechanism that simplifies data retrieval by restricting the search space. A similar filtering capability was incorporated into HIP. NoteCards is an extensible "idea processing environment" that, like HIP, uses typed nodes and links to organize data in various media. NoteCards is integrated into the Xerox Lisp environment much as HIP is integrated into PICASSO. Like Hypercard, the Macintosh-based hypertext tool, HIP defines several user levels that determine what commands are available at a given time. Finally, the Athena Muse system, designed to support curriculum development by MIT faculty, defines a flexible model for incorporating multidimensional information that significantly influenced the hypermedia abstractions and video interface used in HIP.

3. Example HIP Applications

This section presents two examples that illustrate the use of HIP. The first example focuses on HIP's ability to integrate data in a variety of media within a well-organized, consistent framework. The second demonstrates the navigation tools and complexity-reducing features of the system.

3.1 The Engineer's Notebook

The Engineer's Notebook is an application of HIP in the Computer Integrated Manufacturing (CIM) domain [Bec90]. It provides a paperless medium in which process engineers in a microfabrication facility can record data associated with steps in the manufacture of semiconductor chips. Normally, process engineers carry paper notebooks in which they record tables of measurement data, textual comments and observations, and simple line-drawings of wafer profiles or junctions. The Engineer's Notebook captures this same information in an on-line system, as Figure 1 illustrates. This figure presents some typical Notebook components and HIP windows: 1) the top-level control window (upper left); 2) the introductory node of the Notebook, containing a textual outline of the processing steps to be documented (lower left); 3) a notebook entry containing a wafer profile image (lower right); and 4) the overview map of the Notebook hyperdocument (upper right).

This system has several advantages over a traditional paper notebook. First, it reduces the use of paper in the facility, which in turn reduces the particle count in the clean-room environment. Second, it allows the engineer to include many types of data not easily represented in a traditional notebook, such as digitized images and video sequences. Finally, it supports cross-referencing of

information (e.g., process measurements and test data across many runs), both within an individual's notebook and across several notebooks.

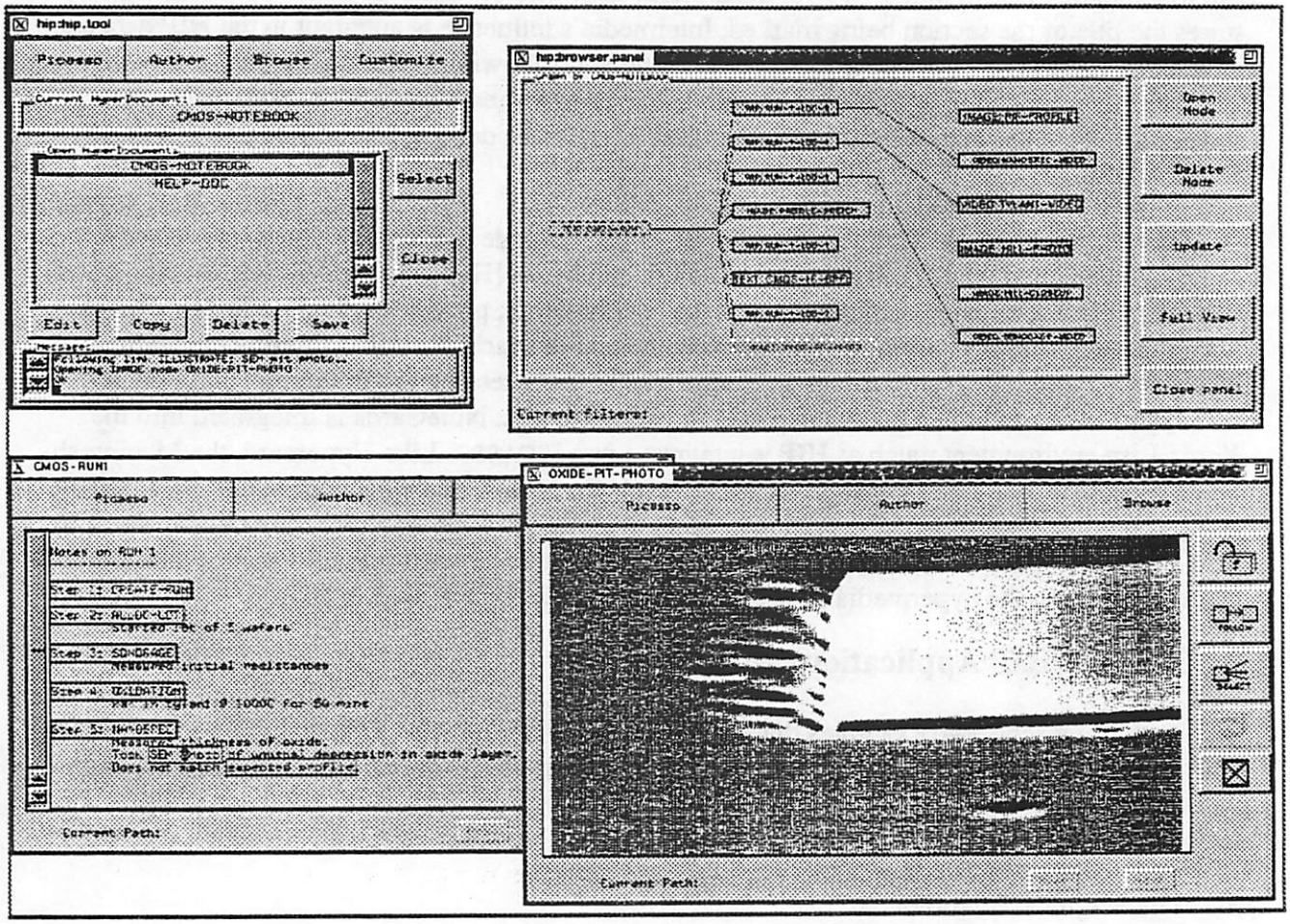


Figure 1: Example HIP application: The Engineer's Notebook

Pictured are (clockwise from upper left): The HIP top-level control panel; a graphical map of the current Notebook; an image node displaying a wafer profile; and a text node outlining processing steps.

A related application will present training materials for users of the Berkeley Microfabrication Facility. HIP is being used to develop a multimedia introduction to integrated circuits, semiconductor design and manufacturing, and the semiconductor industry as a whole. The application will include: 1) textual descriptions of the history of integrated circuits, semiconductors, and manufacturing equipment and processes; 2) video sequences illustrating proper use of equipment; 3) video "tours" of the lab; and 4) cross-referenced instructional materials for each step of a simple process (e.g., step parameters, equipment use, and wafer profiles and pictures showing the desired results of the processing).

As demonstrated by these applications, HIP provides a powerful environment in which multimedia data can be integrated within structured, on-line documents.

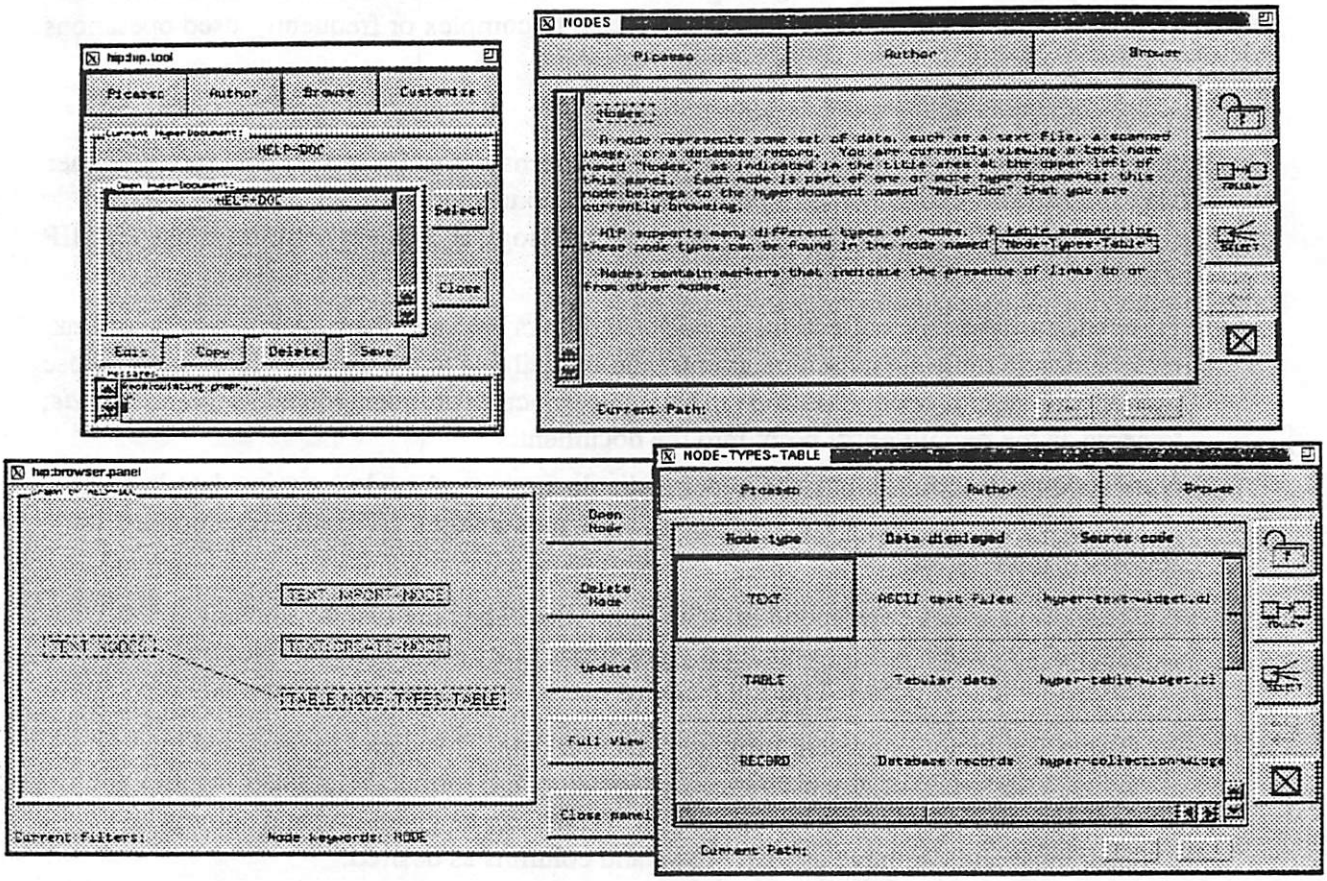


Figure 2: Example HIP application: The HIP Help system.

Pictured are (clockwise from upper left): The HIP top-level control panel; a text node describing what nodes are; a table node summarizing node types; and a graphical display of the hyperdocument, filtered to show only nodes involving the concept "Nodes".

3.2 The HIP Help system

The second example of HIP's features is the system's own on-line Help system, illustrated in Figure 2. Help is presented as a hypermedia document containing information about all objects and operations in HIP. This document is opened by default when a user begins a HIP session. It provides both a starting point for learning to use the system and a handy reference for later questions. The Help document includes two indices: one subject-oriented, like a table of contents, and the other alphabetical, as in a back-of-the-book index.

HIP's navigational aids are employed to make this Help system as easy to use as possible. Link typing is used to distinguish the `Topic` links associated with the subject-oriented index from the `Index` links associated with the alphabetical listing. This distinction allows the user to filter out all links from one index in order to focus on the other view. Consequently, this document can be used as a subject-oriented user's guide or tutorial, as an advanced user's reference manual, or as a combination of both. Several predefined paths through the Help document are also provided, each offering a hands-on introduction to some aspect of the system. For example, one path

provides a basic overview of the concepts embodied in HIP and the paradigm of hyperdocument browsing; another focuses on authoring commands and more advanced features. The user can also leave bookmarks in the Help document, for easy access to complex or frequently-used operations.

4. The HIP Data Model

This section describes the HIP data model and storage mechanisms. Hypertext data models generally involve nodes (also called *cards* or *pages*) logically connected by links to form a directed graph (also called a *web* or *network*). The specific terminology and object definitions used in HIP are as follows:

(1) **Hyperdocuments** are collections of nodes and links between them. A given node or link may belong to multiple hyperdocuments, and is available to the user if at least one of those hyperdocuments is open. One node in each hyperdocument is designated as the *start-node*, to serve as the default entry point into the document.

(2) **Nodes** are the building blocks of hypermedia documents. A node contains data in some medium (e.g., a text file, a video segment, a scanned image). HIP currently supports the following node types:

> Text. Text nodes correspond to ASCII text files. They are displayed using the PICASSO text editor, which provides Emacs-like editing capabilities on a scrollable text buffer. Text file contents can be edited and saved from HIP.
>
> Table. Table nodes present formatted, tabular data. This data can be stored in a file, a database, or any user-specified location and format. PICASSO's scrolling table browser is used for the display. Tables can be given row and column headings, and can have as many rows and columns as desired.
>
> Image. Image nodes contain scanned images in bitmap or GIF format [Com87]. Both black-and-white and color images are supported.
>
> Graphics. Graphic nodes present line-oriented graphics. They are displayed using the PICASSO graphical browser, which provides panning, zooming, and selection capabilities. A PICASSO direct-manipulation graphics editor is under development and will be integrated into HIP.
>
> Object. An object node represents any complex object with heterogenous attributes. Object nodes are often used, for example, to display database records. PICASSO provides an electronic forms editor that is used in HIP to display objects. The type of each object attribute determines how it is displayed.
>
> Video. Video nodes correspond to video sequences stored on interactive videodiscs. PICASSO provides a sophisticated tool for browsing and indexing videodiscs that supports functionality such as variable-speed play, indexed search, and separate control of audio and video channels. HIP uses a simplified version of this tool that provides the basic control operations and the capability to define new entries in an index for a particular videodisc. Users can create links into video nodes that, when followed, will cause the videodisc to advance to the start frame specified by the destination marker. The

video controls can then be used to view the selection (or any other portion of the disc).

Audio. Audio nodes contain digitized voice, music, or other sounds. Audio is currently handled like video, using the two audio tracks on a videodisc to store and access audio data.

Because the various media are represented by different node types in HIP, the terms "node type" and "medium" are often used interchangeably. In reality, a given medium might correspond to a set of node types. For example, we might define several subtypes of the table node type, each representing a different table data storage format (e.g., one for tables stored in a relational database, another for tables stored in flat files). The user, however, is unaware of these underlying differences, so the various subtypes are generally viewed as a single class.

(3) **Links** express typed relationships between nodes. Links are defined between subregions of nodes rather than between the nodes themselves. A given region, called a *marker*, may be the source or destination for any number of links. Each link has a type selected from an extensible taxonomy and a label that communicates to the user the type of information at the link's destination.

HIP provides an initial taxonomy of link types based on that proposed by DeRose [DeR89]. This taxonomy can be extended by any user to define custom link types for a particular application. For example, a group using HIP to support collaborative document development may wish to define specialized link types for various kinds of comments and annotations. The only restrictions regarding the creation of new types are that all type names must be distinct, and only subtypes of existing types are permitted (i.e., all types must descend from the single base type, `Link`).

(4) **Markers** represent selectable components or subregions of a node. A marker may specify a single item (e.g., an object in a graphics node), a collection of items (e.g., a paragraph in a text node) or all items in a node. Marker regions may overlap within a node. All markers are given user-defined labels, which must be unique within a node.

HIP defines two subtypes of marker: **link-markers**, which serve as anchors for links, and **bookmarks**, which provide arbitrary reference points within a hyperdocument.

(5) **Paths** represent sequences of nodes. A path provides a default traversal order for a subset of the nodes in a hyperdocument.

Figure 3 summarizes the HIP data model with a simple example diagram showing three nodes and the links among them. Note that the text node at the left of the figure has two link-markers, pictured as outlined regions. The top link-marker has a single link from it to a link-marker representing a column in the table node on the right. The other link-marker serves as the source for two links, one to the table node and one to the graphics node, and thus is shown with a double outline. Similar visual cues are used in the HIP user interface to convey link and marker information.

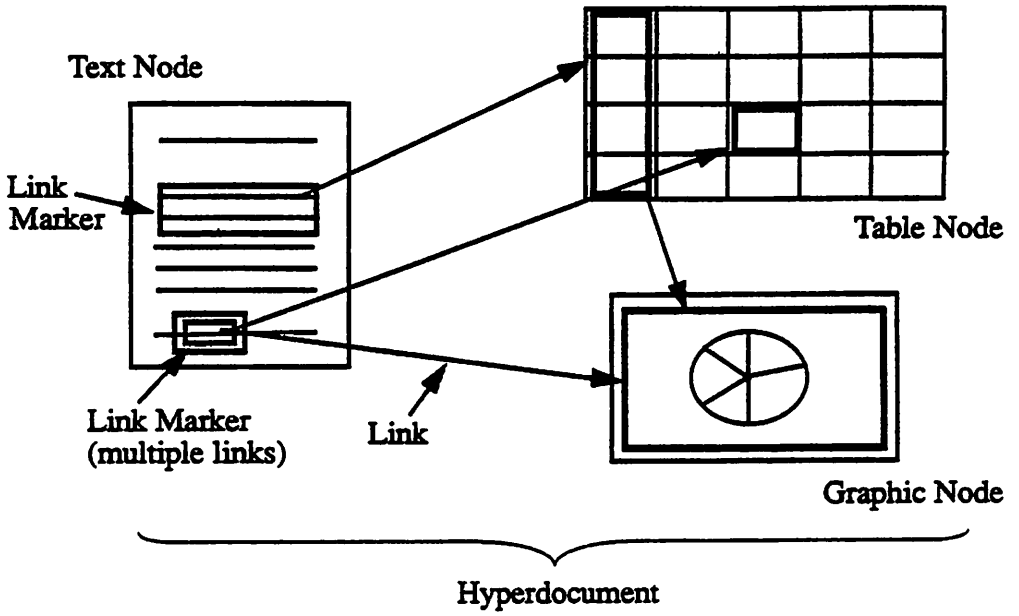


Figure 3: The HIP data model

Nodes, links, and hyperdocuments can all be given arbitrary textual descriptions and keyword attributes. These attributes are used both as documentation and as selection criteria for viewing a subset of the hyperdocument using filtering (see Section 5.5).

HIP supports controlled sharing and collaboration through the use of document ownership and access permission attributes. All HIP documents are owned and can be assigned permissions specifying read, append, or edit access. *Read* access designates a document as read-only. *Append* access allows a user to annotate the document with new nodes and links, but not to modify existing elements or attributes. Finally, *Edit* access permits not only additions to a document but also modifications to its attributes and those of its constituents. At the node level, Append access allows the addition of links to or from a node, while Edit permission allows modification of the actual node contents as well as its attributes.

HIP objects are stored in a relational database [Ing89]. This database system provides the access control and data management facilities needed to support document sharing and searching. The database is used to store the attributes of and relationships among hyperdocuments and their constituent nodes and links. Actual node contents are not stored in the database; instead, a pointer such as a filename, database relation identifier, or videodisk label is stored to indicate the location of the contents. This indirection has several advantages: 1) it allows HIP to use a traditional database instead of requiring a multimedia database; 2) it greatly reduces the size of the database; and 3) it allows users to access hypermedia data outside of HIP and easily import existing data into the system.

HIP also provides an interchange format that allows hyperdocument information to be stored in a file system. Operations for importing and exporting hyperdocument data using this format are

provided. When a hyperdocument is exported, the attributes and link information for each of its nodes is also exported. This process creates a set of files which can then be edited outside of HIP and subsequently imported again, allowing substantive changes to be made with the user's favorite editor. When a node is selected for display, the modification date on its link file can be checked to determine whether that information should be used in place of the current database contents.

5. Using HIP

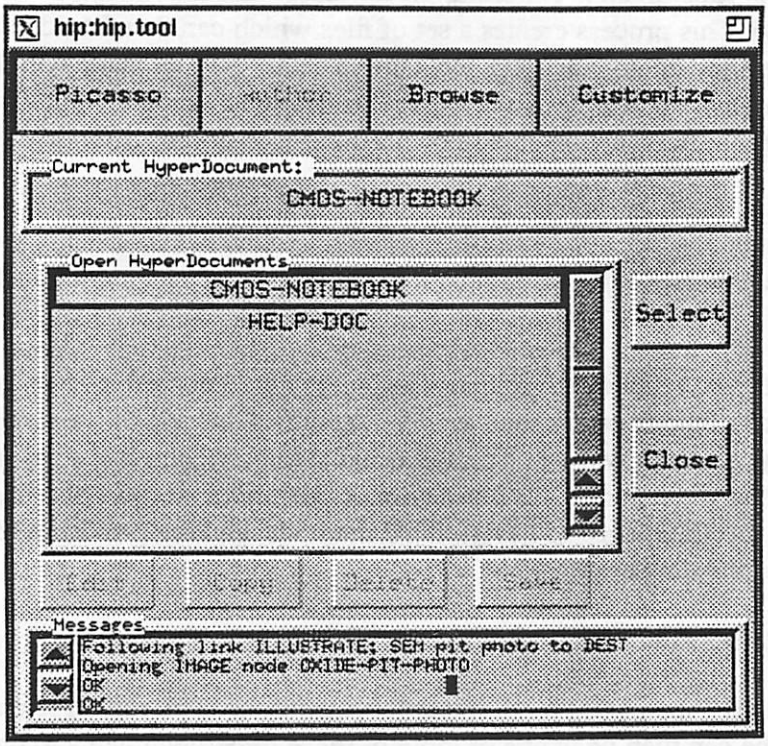
This section describes the HIP interaction model and presents the main components and features of the user interface.

The HIP user interface reflects two primary design goals: 1) consistent treatment of a variety of media, including continuous media such as video, and 2) reduction of the navigational complexity inherent in hypermedia systems. The second goal in particular had a major impact on many aspects of the design, including the placement of operations, the use of modes to distinguish different user levels, the abundance of navigational methods, and the provision of filters to manage the complexity of the data presented.

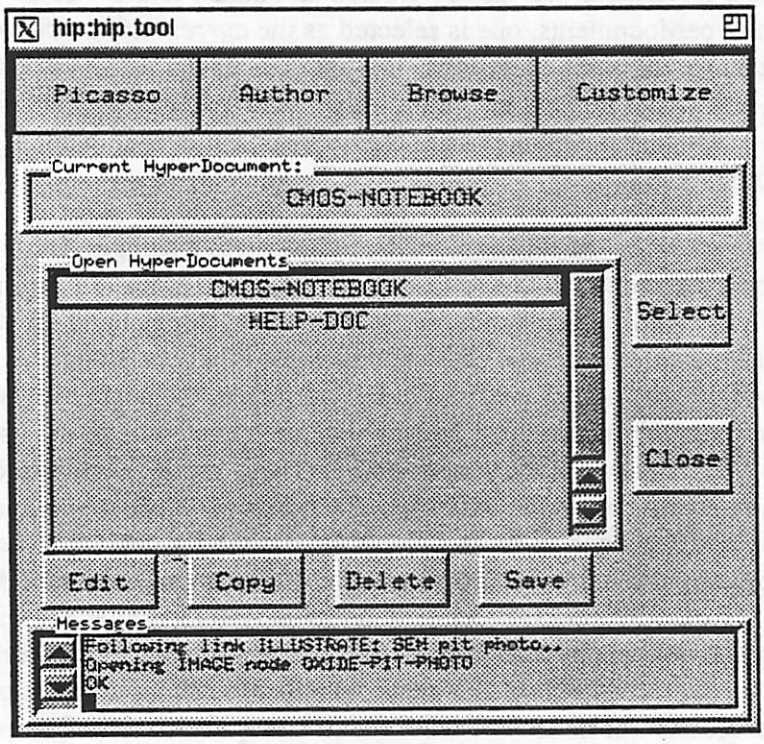
5.1 Beginning a HIP session

When a user invokes HIP, the names of all hyperdocuments she is permitted to access are fetched from the database. She can then *open* one or more of those hyperdocuments to create a working set for the session. New documents can also be created and added to the working set at any time. From this set of open hyperdocuments, one is selected as the *current hyperdocument*, which is then used as the context for all other operations. The user can easily move from one open document to another, as a researcher in a library might utilize several reference books spread out on a table. Bookmark and history list mechanisms are provided to reduce the potential disorientation of having multiple documents in use.

Figure 4 shows two views of the top-level HIP window, with a sample list of open hyperdocuments and the document named "CMOS-NOTEBOOK" selected as the current hyperdocument. These two views show the same window, but in 4(a) the AUTHOR menu and the commands below the list of open hyperdocuments are dimmed, whereas in 4(b) they are selectable. This difference reflects a change in *user mode*. Modes are used in HIP to reduce the cognitive overhead of using the interface by presenting only those commands which are appropriate to the user's task. HIP supports three user modes: *Browse*, *Author*, and *Modify*. These modes correspond to the Read, Append, and Edit permissions defined on HIP objects. In *Browse* mode, shown in 4(a), the user can open and traverse any hyperdocument for which she has read access, and can create bookmarks for use in navigating through those documents; no other creation or modification operations are accessible. Figure 4(b) shows the same window in *Author* mode: commands are available for creating and deleting hyperdocuments, nodes, and links and for editing the attributes of existing components (again subject to user permissions). *Modify* mode additionally allows the user to edit the actual contents of nodes, for those media that provide editing capabilities. The CUSTOMIZE menu provides the USER MODE command for changing the current mode.



(a) Browse mode



(b) Author mode

Figure 4: The top-level HIP window in (a) Browse mode and (b) Author mode

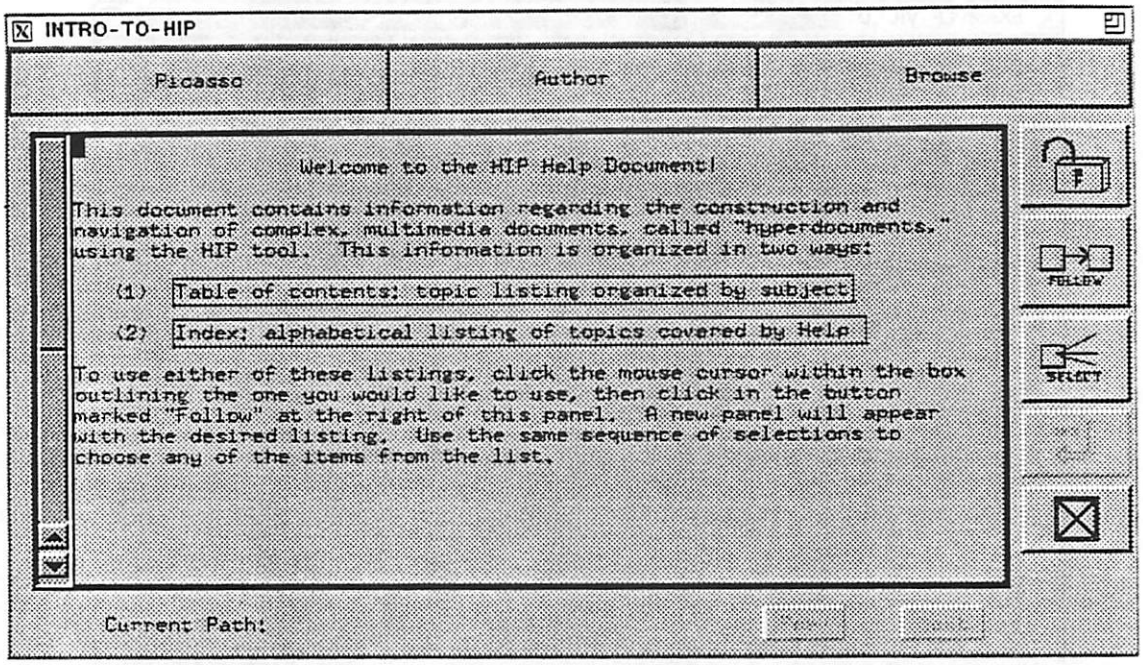


Figure 5: Introductory node of the HIP Help document

5.2 Browsing a hyperdocument

As described above, the user chooses a hyperdocument to browse by selecting a name from the list of open hyperdocuments. When a hyperdocument is selected, it becomes the current hyperdocument and its start-node is automatically opened. The process of opening a node involves fetching the contents of the node and presenting that data in one of the HIP display panels. Figure 5 shows one such panel, showing the start-node of the HIP Help document.

HIP maintains a cache of these panels, allocating them according to a least-recently-used algorithm. When a node is opened, a panel from the cache is selected based on the time each panel was allocated for its current node or the last time an operation was performed from that panel. A panel can be locked, using the lock button shown in the upper-right of the panel in Figure 5. Locked panels will not be reused, allowing frequently-accessed nodes to remain available. The size of the panel cache (initially three) can be adjusted by the user to allow for a larger or smaller working set of nodes. Panels can also be resized and closed as desired.

Video data is also displayed using these panels, but with some additional commands for controlling the videodisc player, as shown in Figure 6. . Since markers denote sequences of frames rather than spatial coordinates, a timeline is used to display marker regions. The buttons under the marker display area bring up control panels for the videodisc player which allow the user to play, scan, search, and create markers in the video. Video data is currently displayed on a monitor adjacent to the workstation, but we recently received a video overlay board that will enable it to be shown in the panel itself. The image shown in the figure is not actual video output, but illustrates how the video will be appear when this board is in use. The video model will also be extended to allow markers to specify regions within frames as well as temporal sequences of frames.

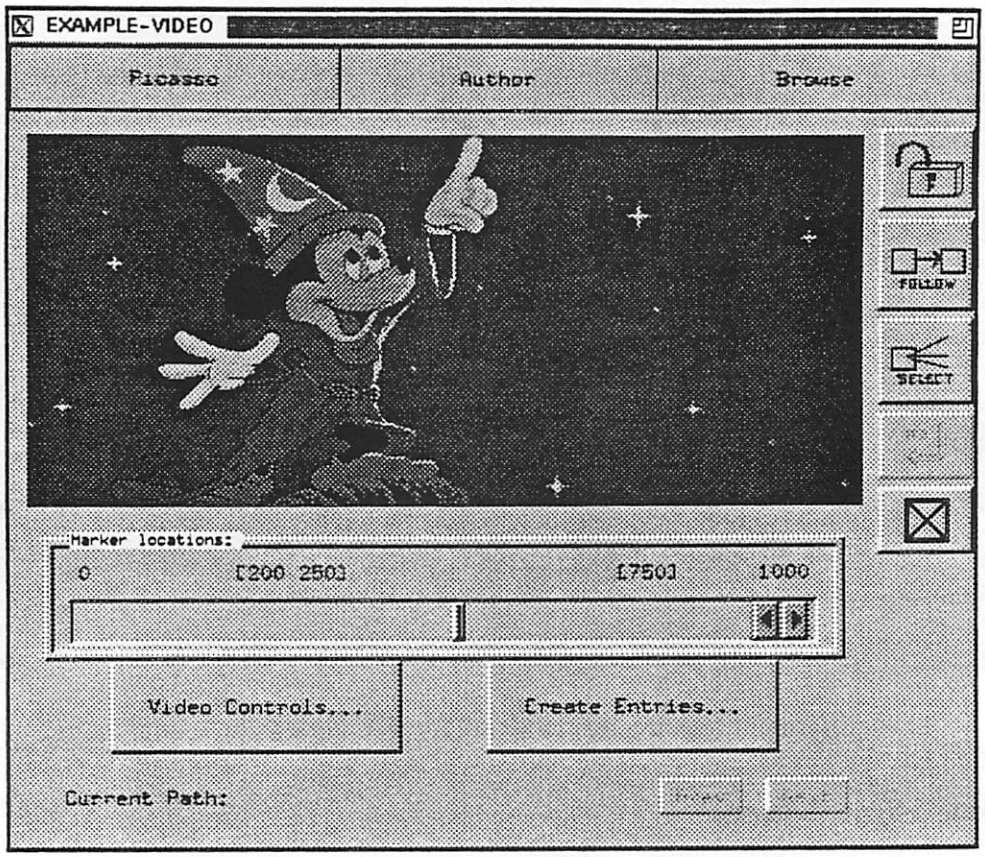


Figure 6: Example video node display

The node shown in Figure 5 contains two link-markers, as indicated by the outlined regions in the text[3]. The single outline around a marker denotes that a single link can be followed from that point. To traverse that link, the user can select the link-marker (by clicking the left-mouse button anywhere in its region) and then select the FOLLOW button at the right of the panel. If a link-marker has multiple outgoing links, indicated by a double border (see Figure 1, lower-left panel), the user can press the button labeled SELECT to bring up a menu of the links associated with the chosen marker. This menu allows the user to examine the type and destination of a link before incurring the overhead of opening its destination node, and thus is useful even with single links. If the FOLLOW button is used with a marker with multiple links, the first link in the set is traversed. Keyboard shorthands are also provided for both of these commands.

Another way to see the links associated with a node is to use the SHOW ALL LINKS command in the BROWSE menu on the panel. This command brings up a menu of all links with a source or destination within the current node. The user can then follow any link directly, without having to locate and select its marker in the node.

3. Marker display can be suppressed and recalled through a command on the BROWSE menu.

Link traversal is the primary means of navigation in most hypertext and hypermedia systems. Because link traversal alone can be disorienting and inefficient, HIP provides a variety of other tools to aid the user in navigating complex documents:

(1) **Node list.** An alphabetical list of the nodes in the current hyperdocument is available via a menu command. This listing includes not only the name of each node, but also its textual description, keywords, and dataset location. This tool is useful for the user who knows the name of a node but not its location within the hyperdocument, or for obtaining a more detailed overview of the nodes in the current hyperdocument.

(2) **Node visit history.** All nodes visited during a session are kept in a history list, which can be called up via a menu command and used to return to a previously-visited node.

(3) **Backtracking.** If the user traverses a link to arrive at a node, the RETURN button on the node panel (represented by a left-pointing arrow icon) is made selectable. The user can select this button to return via that link, backtracking to the source node.

(4) **Bookmarks.** A bookmark can be left in a node by marking a region in the node and choosing the CREATE BOOKMARK command under the BROWSE menu on the node panel. This command prompts the user for a descriptive label for the new bookmark. The SHOW BOOKMARK LIST command under the top-level BROWSE menu provides a menu of these descriptors that allows the user to select a bookmark and return to the specified region. Bookmarks are not saved between HIP sessions.

(5) **Graphical overview map.** The graphical browser presents an overview of the hyperdocument as a directed graph or network of nodes (see Figure 7)[4]. The user can open or delete a node directly from the map, and can easily update the graph as nodes and links are created or deleted[5]. Color is used to distinguish visited nodes, and a dotted outline indicates nodes currently being viewed. This tool is an application of the Picasso graphical browser, which provides panning and zooming functions; the user can thus get a feel for the structure of large hyperdocuments and then zoom in to inspect smaller groups of nodes.

(6) **Paths.** A path is an ordered sequence of nodes in a particular hyperdocument. By choosing one of the paths defined on the current hyperdocument, the user can simply use the PREV and NEXT commands on the node panels to move from node to node along that path. A path can be entered either by using the SELECT PATH command to choose from all available paths, or by selecting the SHOW PATHS command in the BROWSE menu of a node panel. The latter command brings up a menu of all paths through that node and can be used to enter any one of them at that point. A path can be suspended at any point during traversal and resumed from that point the next time that path is selected.

When a path is selected, the panel interface changes to reflect the new context. Figure 8 shows the introductory HIP Help node from Figure 5 after the user selected the path named "HIP-TUTORIAL-1". Note that the name of the path appears at the bottom of the panel, and the PREV and NEXT buttons are selectable while the FOLLOW and SELECT buttons are dimmed. The link-markers in this node are shown as dotted outlines, indicating that their links are not currently accessible.

---

4. Graph layout is computed automatically by the GRAB algorithm [Row87]. The Lisp implementation was done by Ralph Marshall [Mar90].

5. Automatic graph update is straightforward, but adds considerable overhead to creation and deletion operations.

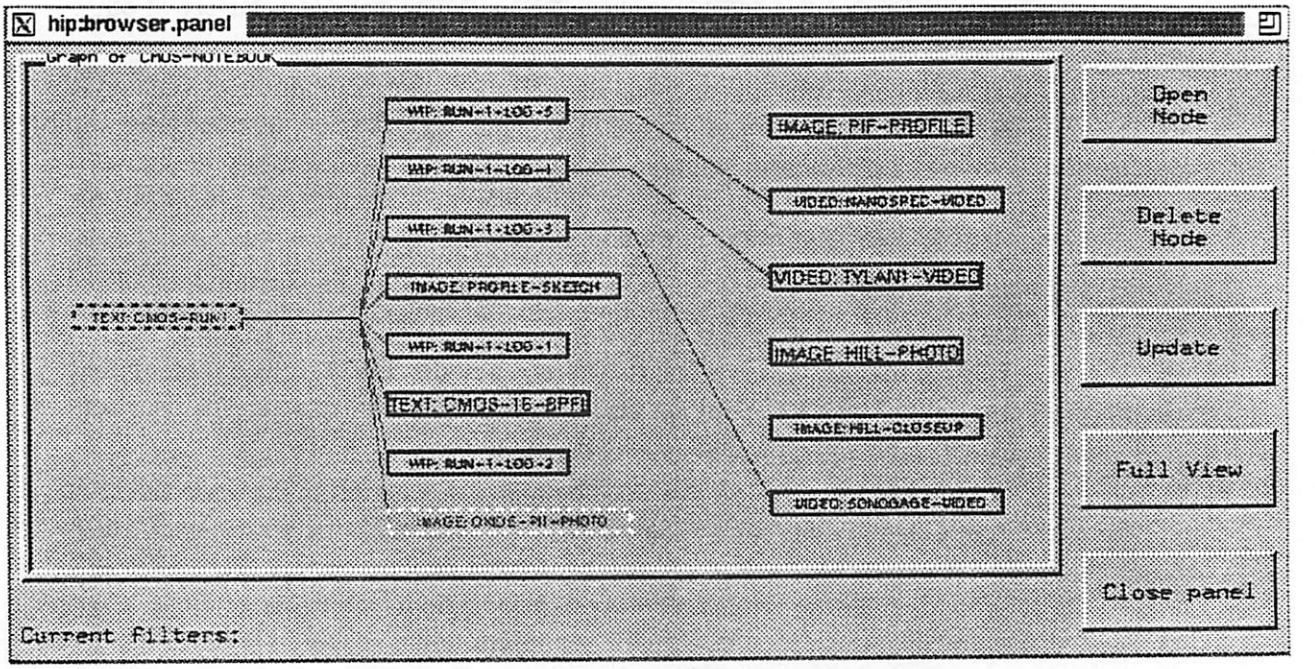


Figure 7: Graphical map of a hyperdocument

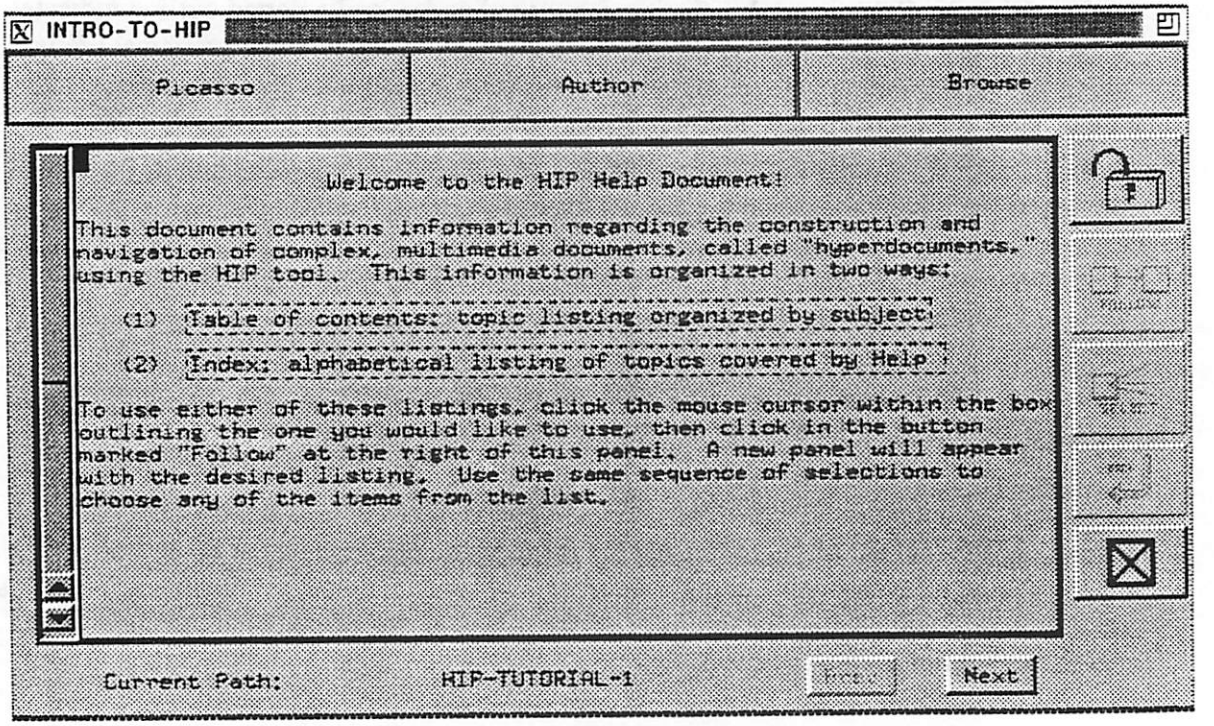
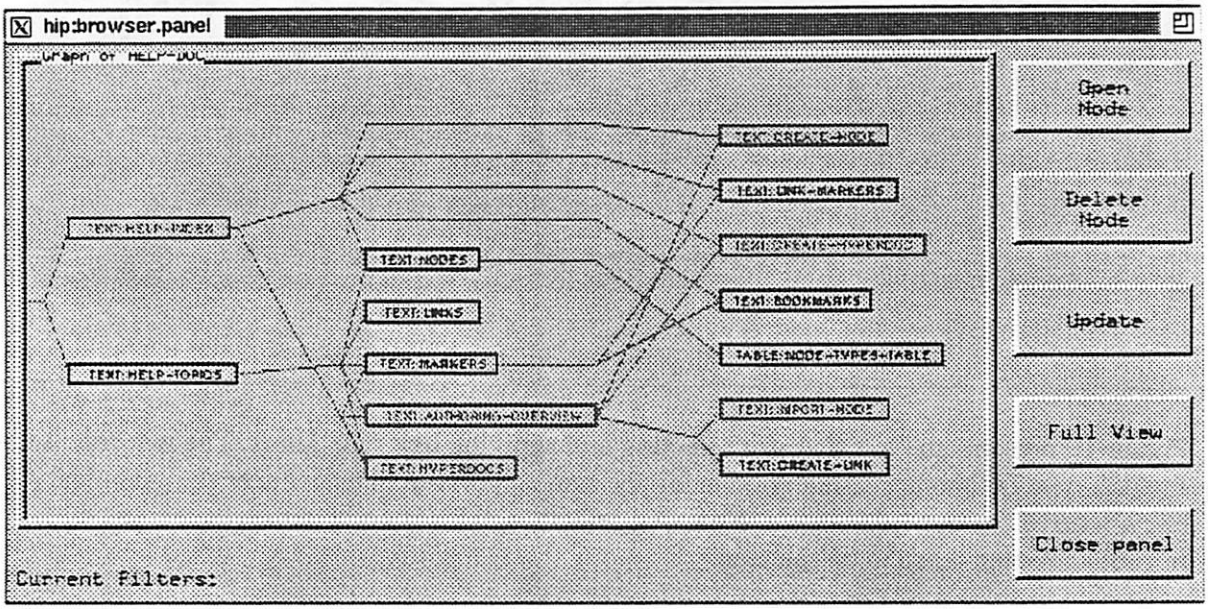
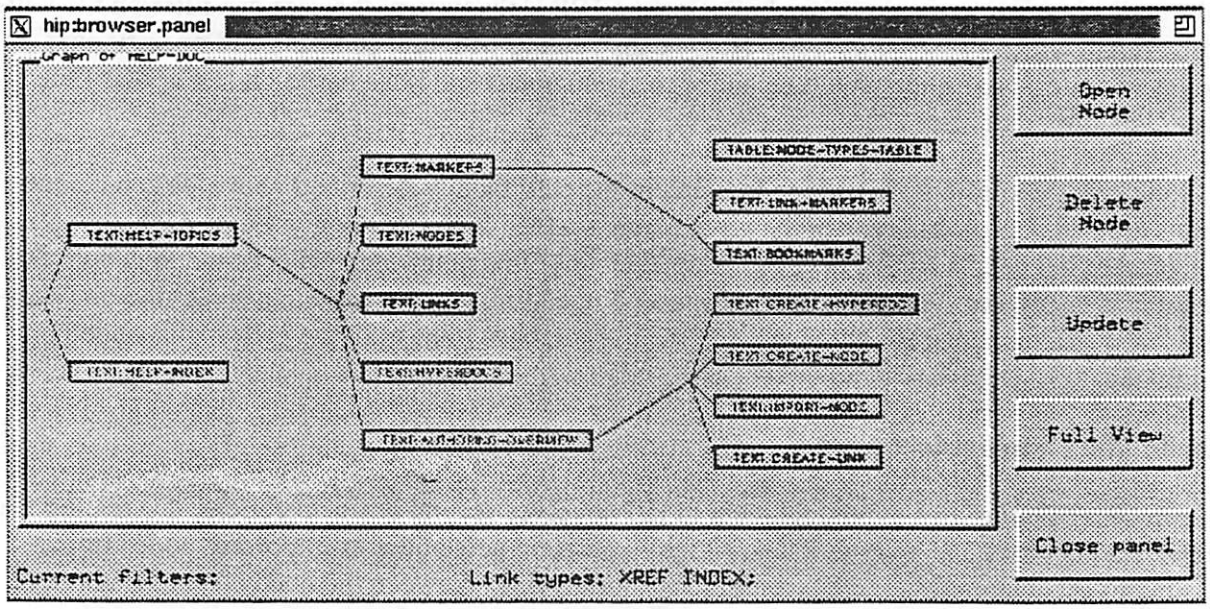


Figure 8: Panel displaying path information and commands

In addition to these navigation tools, HIP provides a flexible filtering mechanism that allows the user to select the amount and type of information presented at a given time. Three filtering criteria are currently supported: link type, node type, and keyword. The link type filter can be used, for example, to show only the `Topic` links in the HIP Help hyperdocument, removing the clutter caused by the `Index` (and other) links. The node type filter works similarly. Figure 9 shows the graphical map of the Help document before and after applying the filtering mechanism: in 9(a), no filters have been applied; in 9(b), link types `Xref` and `Index` have been filtered, as indicated by the filter status line at the bottom of the panel.



(a) Graph of hyperdocument with no filters applied.



(b) Graph of hyperdocument after filtering link types `XREF` and `INDEX`.

Figure 9: The graphical map of the HIP Help document, before and after applying type filters.

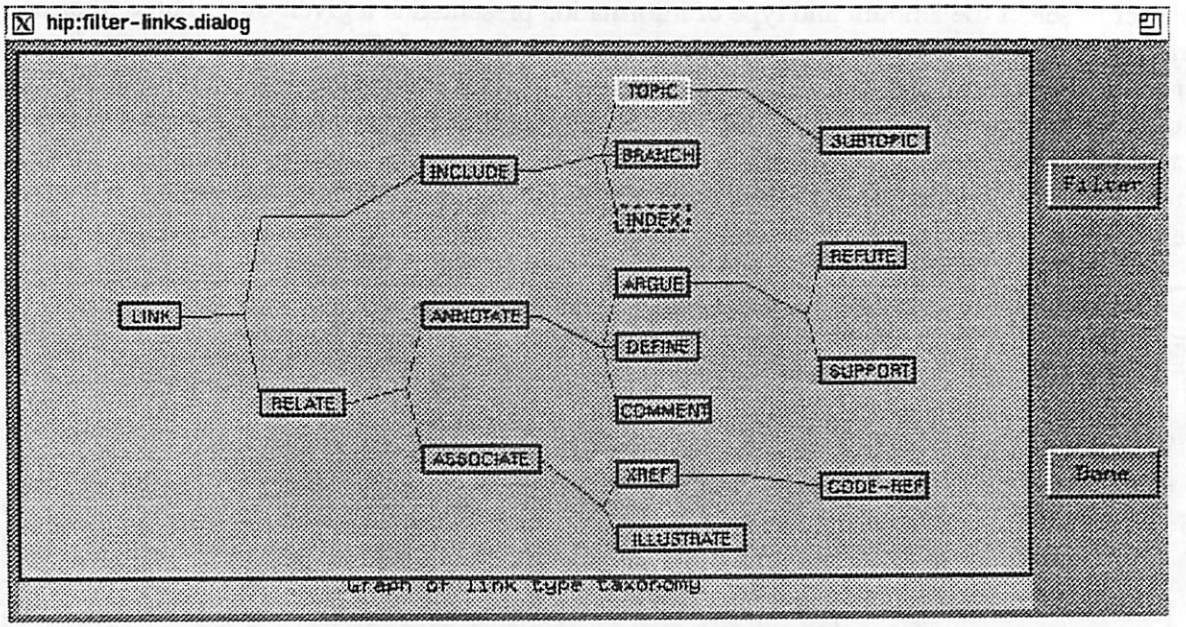


Figure 10: The dialog for selecting link type filters.

Figure 10 presents the interface for selecting and applying link type filters. The tree representing the link type taxonomy is generated using the same algorithm that produces the graphical overview map of the hyperdocument. Note that selecting an inner node in the tree causes the entire subtree rooted there to be selected.

Keyword filters allow the user to select a set of keywords that define a subset of nodes and/or links to display. For example, when browsing the HIP Help system, the user can filter on the keyword "link," which causes only those Help nodes dealing with link concepts and operations to be displayed. Additional filters could be added to select nodes and links based on last access time, owner, creation date, etc.

Link types and filters are used to implement the path facility described above. When a new path is created, a new link type of the same name is added to the taxonomy. Links of that type are then created between adjacent nodes in the path, using "invisible" markers (i.e., markers with null regions) as endpoints. When the user selects a path, all link types are filtered except the one associated with the path, effectively restricting the user to those links until the path is suspended or completed.

5.3 Building a hyperdocument

The process of building a hyperdocument in HIP involves several basic operations: 1) creating the new hyperdocument; 2) adding nodes, either by creating new nodes or importing nodes from existing hyperdocuments; and 3) creating appropriate links between the nodes. Table 1 summarizes the commands provided to support these operations.

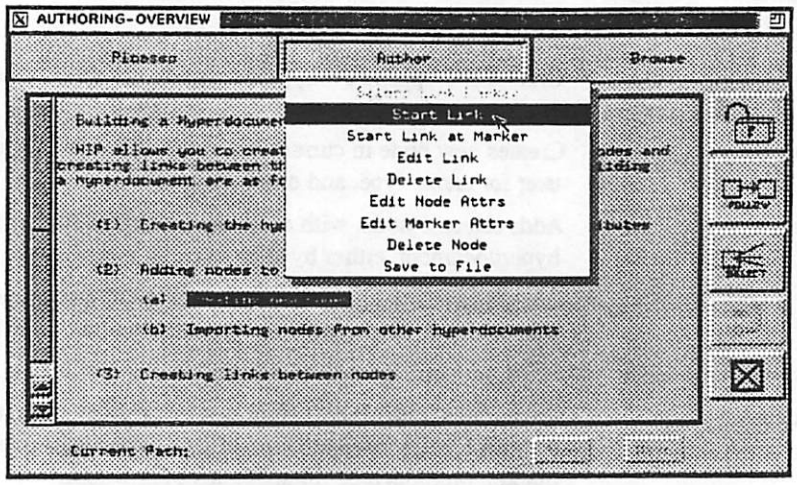
TABLE 1. Operations used in building hyperdocuments

Operation	Description
New Hyperdocument	Creates and opens new hyperdocument, prompting user for name and attributes.
New Node	Creates new node in current hyperdocument, prompting user for name, type, and other attributes.
Import Node	Adds selected node, with or without its links, to current hyperdocument, either by sharing or by copy.
Start Link	Creates new link marker around selected item(s), prompts user for label, starts new link from that marker.
Start Link from Marker	Starts new link from selected marker.
End Link	Creates new link marker around selected item(s), prompts user for label, sets destination of link in progress to new marker, prompts user for attributes of new link.
End Link at Marker	Sets destination of link in progress to selected marker, prompts user for attributes of new link.

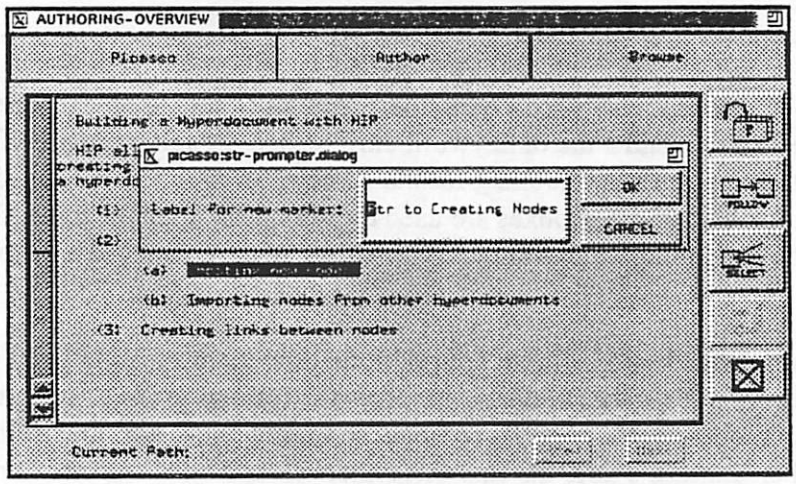
Most of these operations involve specifying the attributes of a new object. These attributes include a textual description and a list of keyword attributes, a default pathname for hyperdocuments (used as a starting point when looking for the contents of its nodes), type and data location for nodes, and type for links. Dialog boxes are used to prompt the user for this information, as illustrated in Figure 12. Similar dialogs are used for subsequent requests to edit the attributes of existing objects.

The last four commands listed in Table 1 support the process of creating links between nodes. This process, illustrated in Figures 11 and 12, involves the following steps: opening the source node, starting the link, opening the destination node, ending the link, and specifying the attributes of the new link. When starting a link, the user can either create a new link-marker implicitly, by marking a region in the node and selecting the START LINK command, or can add the link to an existing link-marker by selecting the desired marker and choosing the START LINK FROM MARKER command[6]. Ending the link is analogous; command names automatically change to END LINK and END LINK AT MARKER, respectively, until the destination is specified. The dialog box of Figure 12 is presented after the link's destination has been chosen to allow the user to set the type and other attributes of the link. The link type can be entered in the Type field directly, or can be selected from a graphical view of the taxonomy by pressing the SELECT TYPE button. This button brings up another dialog box showing the graph of the taxonomy (similar to the dialog for choosing link type filters), from which the user can select an existing type or can add a new subtype to be used for the link.

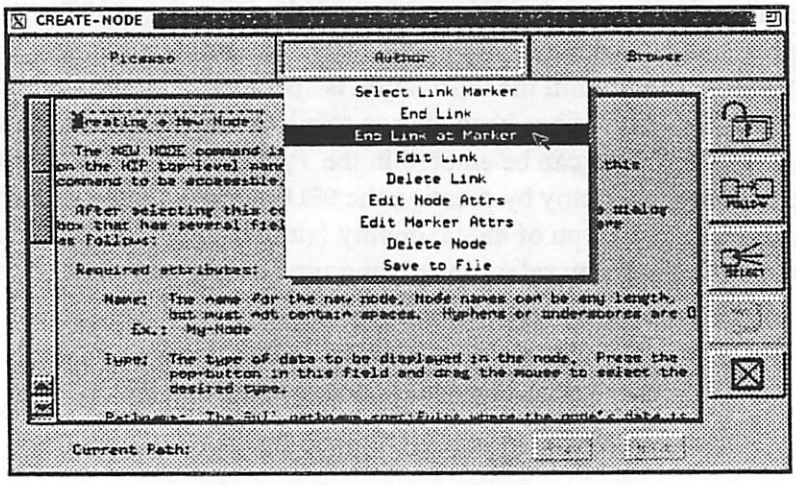
6. Originally, a single START LINK command was used, internally selecting the appropriate option according to the current mark region or cursor position; however, since marker regions can overlap or can be created around single points, the commands were separated in order to remove ambiguity.



(a) Staring a link from a marked region of text



(b) Labelling the new marker



(c) Ending the link at an existing marker

Figure 11: Steps involved in creating a new link

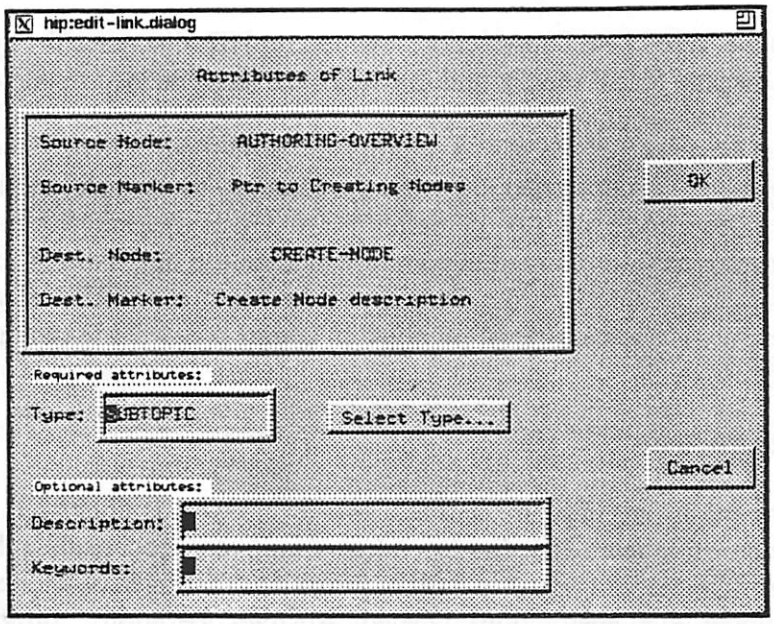


Figure 12: Specifying the attributes of the new link

In addition to these commands for creating and editing hyperdocument, nodes, and links, HIP provides authoring operations for creating paths through hyperdocuments. The NEW PATH command under the top-level AUTHOR menu brings up the dialog box shown in Figure 13. This dialog allows the user to build up a path through the current hyperdocument by selecting nodes from a graphical representation of the document.

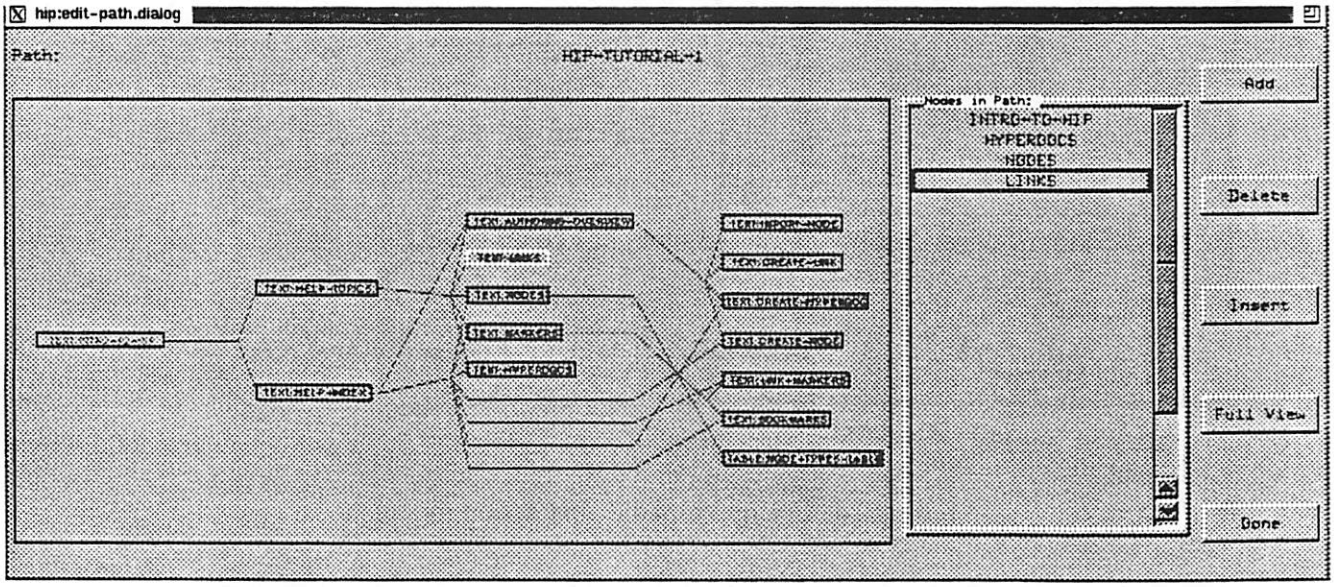


Figure 13: Creating a new path through the HIP Help document

6. Extending Picasso to Support Hypermedia

This section outlines the hypermedia abstractions defined for HIP, discusses the approach taken to provide a consistent display for all media, and illustrates the steps needed to add a new node type.

Many hypermedia systems include custom editors and browsers for text, video, timelines, and other media. HIP, however, was designed to extend the PICASSO toolkit, and therefore utilizes the underlying PICASSO editors and browsers (hereafter referred to as *widgets*) to present all media. A set of hypermedia primitives were defined that are overlaid on top of existing Picasso widgets to provide a consistent interface across media. This approach also makes HIP easily extensible to new media, since the only significant effort involved in adding a new node type is to implement the necessary primitives.

6.1 Hypermedia abstractions

The hypermedia abstractions are implemented as a two-level layer, as illustrated in Figure 14(a). The low-level abstractions, labeled "Media-dependent primitives," encapsulate the interface between HIP and the PICASSO widget set in a set of primitives that must be implemented for each type of widget. The upper level abstractions, labeled "Abstract operations," provide high-level operations defined in terms of those primitives. These operations are in turn called by the HIP interface to implement generic facilities for displaying nodes and manipulating link and marker information.

The hypermedia abstraction layer is specified in a "mixin" class[7]. As shown in Figure 14(b), this class is combined with each PICASSO widget class to derive a set of "hyper-widget" classes. Instances of these classes are then created to display nodes in the interface panels.

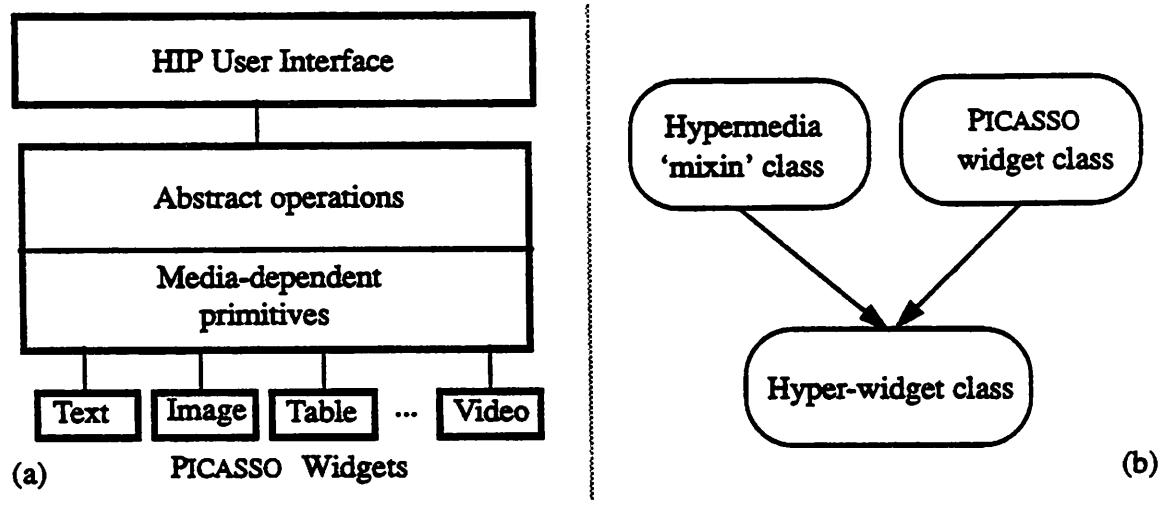


Figure 14: The hypermedia abstraction model

7. A "mixin" class is one whch is used only in conjunction with other classes, to add characteristics or functionality.

TABLE 2a. Media-dependent primitives

Method	Action/Return value
`Widget`	Actual Picasso widget that does I/O
`Setup-node`	Access and load node data
`Get-current-position`	Current cursor coordinates
`Get-mark-region`	List of coordinates describing currently selected region
`Marker-outline-region`	List of coordinates to use to outline a given marker
`Translate-point`	Translate abstract coordinates to screen coordinates
`Scroll-to`	Scroll widget to specified coordinates

TABLE 2b. Abstract operations

Method	Action/Return value
`Get-current-marker`	Marker indicated by current cursor position
`Select-marker`	Identify particular marker as current
`Draw-marker`	Create marker outline, using `translate-point`
`Do-repaint`	PICASSO repaint method, specialized to draw markers

Tables 2a and 2b summarize the methods defined for the two abstraction levels. The media-dependent primitives in Table 2a are specialized for each class of hyper-widget. These functions are primarily concerned with isolating the details of the widget coordinate system. The abstract operations shown in Table 2b deal mainly with marker access and display, relying on the underlying primitives to obtain cursor position information and to calculate and display marker regions.

As noted above, a primary goal of the abstraction mechanism is to encapsulate the widget coordinate system, allowing marker regions to be treated in higher levels simply as lists of tuples. These tuples, which may represent row/column pairs for tables or text, x/y coordinates for images, or frame numbers for video, are returned by the widget (via the `marker-outline-region` method) when the marker is created. When a marker is to be displayed, its region is translated into window coordinates, again at the lowest abstraction level. This approach provides great flexibility in handling media of arbitrary dimensionality and potentially inconsistent coordinate systems.

6.2 Handling multimedia display consistently

Given the hypermedia abstractions described above, the next task was to determine how to display the various media using PICASSO interface objects. One option was to define a specialized type of display panel for each medium; however, that approach would significantly complicate the task of extending the HIP framework for new media, and would lead to an explosion in the number of panels needed. Instead, HIP provides a single type of display panel that presents all hypermedia operations consistently, regardless of node type, while retaining the full functionality of the underlying medium.

This generality was achieved by associating with each panel an instance of each type of hyper-widget. The appropriate hyper-widget to use is determined at run-time by the type of node being

displayed. The `setup-node` method for that hyper-widget class is then called to initialize it with the given node[8].

One disadvantage of using a generic panel was that we were unable to take full advantage of PICASSO's powerful constraint mechanism, which can be used to propagate changes from one variable to another [Row90]. We had originally hoped to bind the availability of various operations to the state of the currently-viewed hyper-widget. For example, the current cursor position would dictate whether the link-following commands were dimmed or not, depending upon whether that position was over a link marker. However, the inconsistency of both attributes and widget behavior across the different widgets made this approach unwieldy. To avoid adding undue complexity to the implementation (and reducing extensibility), we instead provide textual feedback when an operation is selected in an inappropriate state.

6.3 Extensibility

As described above, media are represented in HIP by hybrid widget classes that inherit hypermedia functionality from an abstract class. Adding a new medium thus involves deriving a new hyper-widget class using this mixin. In addition, a new node type must be defined to represent the new medium in the data model. The following example illustrates how animations would be integrated into the HIP framework:

(1) Define a new subclass of node, named `animation-node`, to represent the new media type. Any necessary initialization or saving procedures must also be defined by specializing the `new-instance` and `save` methods on the new type.

(2) Define a new hybrid widget class, `hyper-animation-widget`, inheriting from both the `animation-widget` class and the `hypermedia-mixin` class. Using the primitives supplied with the animation widget, specialize the hypermedia abstraction. In particular, define the procedures for loading an animation, displaying marker regions in an animation, and returning current position and region information.

(3) Allow `animation-nodes` to be displayed by installing `hyper-animation-widgets` in the node display panels. This step involves making minor additions to the code used to generate those panels, essentially giving each panel a new child[9].

Extending the system to support a variation on an existing medium involves only a subset of these steps. For example, the Engineer's Notebook application required the presentation of work in progress (WIP) database records. To accommodate this new data type, a new node type, `WIP-node`, was defined as a subtype of `object-node`. A `setup-node` method was then defined for loading the database information into the format required by the object display widget.

---

8. For efficiency, we plan to modify this approach to use caches of each kind of hyper-widget, filling each cache only when a node of its type is opened. Panels will still be generic, and will be assigned a hyper-widget from the appropriate cache when allocated for a particular node.

9. Again, this will change: instead of modifying the panel code, this step will involve defining a method that, given an animation-node, returns an instance of hyper-animation-widget.

TABLE 3. Breakdown of HIP code by component

Component	Lines of Common Lisp code (approximate)
Data model	1500
Hypermedia abstractions	3500
User Interface	3000
Interface support	3000
Total:	11,000

7. Experience

This section describes the lessons learned while designing and implementing HIP. HIP was developed in approximately 10 person-months, and is composed of approximately 11,000 lines of Common Lisp code. Table 3 shows the breakdown of code by component.

The original development approach was to begin with a text-only prototype and then generalize the system to support additional media. It became clear that this approach was inappropriate as we realized that the attributes and characteristics of text do not translate naturally to other media, especially those of different dimensionality. In particular, the characteristics of continuous media such as video are inherently different from those of a static medium such as text. Thus, our attempt to force a textual model of the world onto all other media was soon revised, and led to the development of the hypermedia abstractions described in the previous section.

The desire to provide a consistent interface for all media led to some sacrifice of sophistication. As discussed in 6.2, the use of a generic panel for node display forced us to abandon the PICASSO constraint mechanism as an active feedback mechanism. As the number of media grew, the constraint specification code became unwieldy, since each underlying widget had different attributes from which to propagate changes. To propagate cursor position changes, for example, we would need to define constraints on the `row` and `column` attributes for text, on the `current-indices` attribute for tables, and on the `selection` attribute for graphics, to name a few. Had the PICASSO widget set been designed with a more consistent set of accessors, the HIP implementation at this level could have been considerably cleaner.

Other inconsistencies in the implementation of PICASSO widgets necessitated the use of certain inelegant approaches in HIP. For example, each widget class has a specialized repaint function, normally invoked by the `do-repaint` method. In some cases, however, the internal repaint function is called from other places, such as in the scrolling functions for text-widgets. Consequently, we had to modify some existing PICASSO code to ensure that markers would always be displayed following a repaint operation.

Despite these difficulties, PICASSO proved to be a powerful, useful tool for building what became a fairly complex application. The interface toolkit made the creation of frames and dialogs relatively painless. Moreover, the object-oriented approach used in PICASSO greatly facilitated the addition of a hypermedia framework, since the new "hyper" classes could easily take advantage of the attributes and functionality of existing widget classes. The implementation of the hypermedia abstraction layer in particular depends upon object-oriented techniques, using generic primitives that are specialized for each widget class and used as building blocks for higher levels.

Also, PICASSO's constraint mechanism, though not used as extensively as had been planned, was used to great advantage throughout HIP in maintaining relationships between objects and implementing the more sophisticated browsing capabilities (for example, constraints are used to propagate changes in user mode to the availability of commands in the display panels).

8. Conclusions

The goal of the HIP project was to provide an extensible, easy-to-use hypermedia system using the tools and interface support provided by the PICASSO application development system. HIP has achieved this goal, integrating text, images, tables, graphics, audio, and video within a consistent, extensible hypermedia framework. It provides a wealth of navigation tools and other features designed to reduce the complexity of browsing large hyperdocuments. HIP thus supports a wide range of user needs and levels, from first-time browsers to experienced authors.

HIP is now being used to build applications, including the Engineer's Notebook and semiconductor tutorial examples described in this paper.

Acknowledgments

The author gratefully acknowledges the support and contributions provided by the PICASSO research group members, all of whom graciously responded to numerous requests for bug fixes, new capabilities, and explanations of "undocumented features": Joe Konstan, Chung Liu, Steve Seitz, and especially Brian Smith (whose uniformly excellent ideas are reflected in many of the better aspects of HIP's design). David Mudie was instrumental in implementing the interface between HIP and Ingres, and played a large role in developing the Engineer's Notebook application. Dan Rice participated in the design of the first version of HIP, particularly in working out the details of the data model. Ralph Marshall (MITRE, Bedford MA) generously shared his elegant implementation of the directed graph display algorithm used in the graphical hyperdocument browser and in several other HIP tools.

References

[App87] Apple Computer, Inc., Hypercard User's Guide, 1987.

[Bec90] Becker, B.S., and D. Mudie, "A Paper-Free Replacement for the Engineer's Laboratory Notebook," presented at the 1990 SRC/DARPA IC-CIM Workshop, U.C. Berkeley, August 16. 1990.

[Com87] CompuServe, Inc., Graphics Interchange Format (GIF) Specification, 1987.

[Con87] Conklin, E. J., "Hypertext: An Introduction and Survey," *Computer*, Vol. 20, No. 9 (Sept. 1987), pp. 17-41.

[DeR89] DeRose, S. J., "Expanding the Notion of Links," *Hypertext '89 Proceedings*, Pittsburgh, PA, Nov. 1989.

[Hal87] Halasz, F.G., et al., "NoteCards in a Nutshell," *Proc. of the ACM CHI+GI 1987 Conference*, Toronto, Canada, April 1987.

[Hod89] Hodges, M.E., R. M. Sasnett, and M. S. Ackerman, "A Construction Set for Multimedia Applications," *IEEE Software*, Vol. 6, No. 1 (January 1989), pp. 37-43.

[Hod90] Hodges, M.E., R. M. Sasnett, and V. J. Harward, "Musings on Multimedia," *Unix Review*, Vol. 8, No.2, pp. 83-87.

[Ing89] Ingres Corp., Introducing INGRES for the Unix and VMS Operating Systems, (Release 6.2), June 1989.

[Kee88] Keene, S., *Object-Oriented Programming in Common Lisp*, Addison-Wesley, 1988.

[Mar90] Marshall, R. Implementation of the GRAB directed graph layout algorithm. Personal communication, April 1990.

[Mey86] Meyrowitz, N., "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework," *Proc. OOPSLA '86*, Portland, OR, Sept. 1986.

[Row87] Rowe, L.A., et. al, "A Browser for Directed Graphs," *Software Practice and Experience*, Vol. 17, No. 1 (January 1987).

[Row90] Rowe, L. A., et. al, "The Picasso Application Framework," UCB/ERL M90/18, Computer Science Division -- EECS, U.C. Berkeley, March 1990.

[Shi89] Shipman, F. M., R.J. Chaney, and G. A. Gorry, "Distributed Hypertext for Collaborative Research: The Virtual Notebook System," *Hypertext '89 Proceedings*, Pittsburgh, PA, Nov. 1989.

[Wal85] Walker, J.H., "The Document Examiner," *SIGGRAPH Video Review*, Edited Compilation from *CHI'85: Human Factors in Computing Systems*, 1985.