Intermedia: The Concept and the Construction of a Seamless Information Environment

Nicole Yankelovich, Bernard J. Haan, Norman K. Meyrowitz, and Steven M. Drucker
Brown University

[illegible]

This multi-application hypermedia system provides linking capabilities integrated into a desktop user environment. To promote consistency, the applications were built with an object-oriented framework.

Intermedia, a tool designed to support both teaching and research in a university environment, contains multiple applications and mechanisms to link the contents of documents created with those applications. The system was developed at Brown University's Institute for Research in Information and Scholarship (IRIS).

A hypermedia system expressly developed for use in a university setting, Intermedia provides a framework for object-oriented, direct manipulation editors and applications. With it, instructors can construct exploratory environments for their students as well as use applications for day-to-day work, research, and writing. Intermedia is also an environment in which programmers can develop consistent applications, using object-oriented programming techniques and reusable building blocks.

Hypertext and hypermedia. Although only recently popularized by products like Apple's Hypercard and Owl's Guide, hypertext and hypermedia have been the subject of research, writing, and experimentation for more than 20 years. (Examples of early hypertext systems and existing hypermedia systems may be found in Conklin[1] and Yankelovich.[2]) Intermedia is a direct descendent of ideas developed by such prominent researchers as Theodor Nelson, Douglas Engelbart, and Andries van Dam. Nelson coined the term *hypertext* in the early 1960s to describe the idea of "non-sequential writing." He expanded on that theme in a book he wrote entitled *Literary Machines.*

In essence, a hypertext system allows authors or groups of authors to link information together, create paths through a body of related material, annotate existing texts, and create notes that direct readers to either bibliographic data or the body of the referenced text. Using a computer-based hypertext system, students and researchers can quickly follow trails of footnotes and related materials without losing their original context; thus, they are not obliged to search through library stacks to look up referenced books and articles. Explicit connections—links—allow readers to travel from one document to another, effectively automating the process of following references in an encyclopedia. In addition, hypertext systems that support multiple users allow researchers, professors, and students to communicate and collaborate with one another within the context of a body of scholarly material.

Hypermedia is simply an extension of hypertext that incorporates other media in addition to text. With a hypermedia system, authors can create a linked body of material that includes text, static graphics, animated graphics, video, and sound.

0018-9162/88/0100-0081$01.00 ©1988 IEEE

Intermedia: the concepts

Intermedia is both an author's tool and a reader's tool. The system, in fact, makes no distinction between types of users, provided they have appropriate access rights to the material they wish to edit, explore, or annotate. Creating new materials and making and following links are all integrated into a single seamless, multiuser environment.

Applications. The system, which runs on a network of Unix-based workstations, currently contains five integrated applications: a text editor (InterText), a graphics editor (InterDraw), a scanned image viewer (InterPix), a three-dimensional object viewer (InterSpect), and a timeline editor (InterVal). These applications conform to Macintosh/Microsoft Windows interface standards. Any number of documents of different types, along with the folders containing these documents, may be open on the desktop at one time.

The InterText word processing application resembles Apple's MacWrite, with the addition of style sheets for formatting rather than MacWrite style rulers. Using style sheets, the user can define a set of styles for a particular document (such as paragraph, title, subtitle, indented quote, and numbered point) and apply those styles to an *entity*—the text contained between two carriage returns.

When the user edits the definition of a style, all the entities to which that style are applied reformat accordingly.

With InterDraw, a structured graphics editor similar to Apple's MacDraw, users can create two-dimensional illustrations by selecting tools from a palette attached to each InterDraw window.

InterPix displays bit-map images entered into the system using a digitizing scanner. These images can be cropped, copied, and pasted into InterDraw documents.

The InterSpect viewer converts files containing three-dimensional data points into three-dimensional representations of that data. Users can manipulate three-dimensional images of cells, for example, by rotating them, zooming in or out, or hiding parts of the model.

The fifth application, InterVal, provides interactive editing features for creating chronological timelines. As the user enters pairs of dates and labels, the application formats them on a vertical timeline according to user-defined styles. Like a charting package, the display of the data is determined by a modifiable set of parameters.

Figure 1 illustrates an example of materials from a linked Intermedia *corpus* (collection of documents) called *Context 32: A Web of English Literature*, designed by Brown University English professor George Landow.

User interface. Several user-interface concepts stressed throughout Intermedia enable users to learn new applications quickly and predict the behavior of features they have never used before. In a system that encourages rapid transitions between applications, it is essential to limit the amount a browser must learn in order to successfully use the system and capitalize on those conventions with which he or she may be already familiar. Like the copy and paste operations in Macintosh and Smalltalk programs, some operations in the Intermedia system behave identically across all applications. The linking functionality described below is a prime example.

All applications also provide direct manipulation interfaces. To change the displayed information, the user first selects one or more of the displayed objects and then issues a command through a keyboard or menu interface. Likewise, other system features, while not exactly identical to one another, are conceptually similar. Most applications, for instance, allow users to control the format or the display characteristics of data. We designed the interface techniques for conceptually similar operations to capitalize on the likenesses.

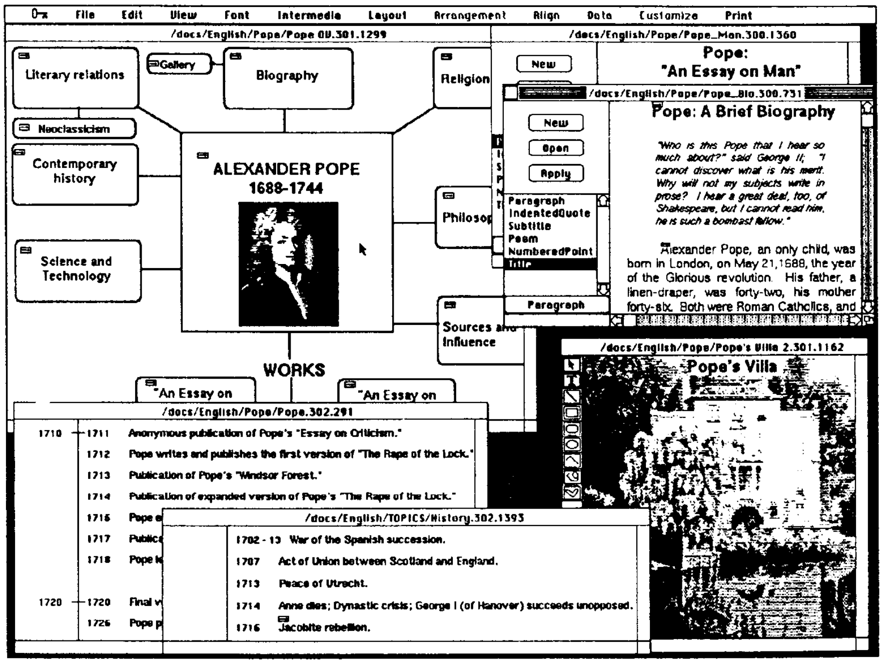
The style paradigm[3] is one example. Styles are sets of properties or characteristics that govern the appearance of data within a document. Users can define or modify a style by editing a form called a style sheet (sometimes referred to as a property sheet). Both the InterText application and the InterVal application contain style sheets to specify different text formats such as paragraphs, indented quotes, lists, and titles, or different timeline formats such as the position of dates relative to tick marks and the position of labels relative to dates. In the graphics editor, different styles may be applied to shapes such as line width, pen style, or fill style. By storing all presentation parameters in style sheets, you can substitute styles with the same name but different parameters for all types of data in the system.

Related to the use of styles is the frequent use of *palettes*—sets of controls attached as a pane to a document window. Along with style sheet dialogs, palettes provide a means for defining and applying styles. In InterText, for example, all the styles defined for a particular document are viewed in a style palette (see Figure 1). Two mouse clicks will change the style of an existing text entity or change the style from one style to another before beginning a new entity. With a large screen and the capacity for 30 or 40 open windows at one time, it is essential that all the tools needed for common operations be close at hand rather than in the pull-down menus. When not needed, all palettes can be hidden from view to unclutter the screen and improve the way material is presented to a person browsing through the system.

The use of "infinite" undo and redo commands—made possible in a workstation environment with virtual memory capabilities—provides another example of a standard user-interface concept that permeates the system. Instead of retracting only the last action performed, the user can incrementally undo the effects of all actions performed since the last time a document was saved. Any single action or set of actions the user has undone can then be incrementally redone. This capability fosters a sense of security in users and enables them to experiment freely with their documents.

Hypermedia functionality. In Intermedia, the hypermedia functionality is integrated into each application so that the actions of creating and traversing links can be interspersed with the actions of creating and editing documents. (The screens in the "Sample session" illustrate the operation of Intermedia, highlighting the hypermedia functionality.)

In an effort to fit the link-making process into a conceptual model already familiar to users, the act of making links between Intermedia documents was modeled as closely as possible on the Smalltalk/Macintosh copy/paste paradigm.[4] If links are to be made frequently, they must be a seamless part of the user interface. In any document, users can specify a selection region and choose the Start Link command from the menu. In any other document, regardless of type, users can define another selection region and choose one of the Complete Link commands.


File Edit View Font Intermedia Layout Arrangement Align Data Customize Print
/docs/English/Pope/Pope OV.301.1299
Literary relations
Gallery
Biography
Religion
Neoclassicism
Contemporary history
ALEXANDER POPE 1688-1744
Science and Technology
Philosop
Sources and Influence
WORKS
"An Essay on
"An Essay on
/docs/English/Pope/Pope_Man.300.1360
New
Pope: "An Essay on Man"
/docs/English/Pope/Pope_Bio.300.731
New
Open
Apply
Paragraph
IndentedQuote
Subtitle
Poem
NumberedPoint
Title
Paragraph
/docs/English/Pope/Pope's Villa 2.301.1162
Pope's Villa
/docs/English/Pope/Pope.302.291
/docs/English/TOPICS/History.302.1393

Pope: A Brief Biography

"Who is this Pope that I hear so much about?" said George II; "I cannot discover what is his merit. Why will not my subjects write in prose? I hear a great deal, too, of Shakespeare, but I cannot read him, he is such a bombast fellow."

Alexander Pope, an only child, was born in London, on May 21,1688, the year of the Glorious revolution. His father, a linen-draper, was forty-two, his mother forty-six. Both were Roman Catholics, and

```
1710 —1711  Anonymous publication of Pope's "Essay on Criticism."
      1712  Pope writes and publishes the first version of "The Rape of the Lock."
      1713  Publication of Pope's "Windsor Forest."
      1714  Publication of expanded version of Pope's "The Rape of the Lock."
      1716  Pope e
      1717  Public
      1718  Pope le

1720 —1720  Final v
      1726  Pope p
```

```
1702 - 13  War of the Spanish succession.
1707       Act of Union between Scotland and England.
1713       Peace of Utrecht.
1714       Anne dies; Dynastic crisis; George I (of Hanover) succeeds unopposed.
1716       Jacobite rebellion.
```

Figure 1. Two InterText documents (top right), two InterVal documents (bottom left), and two InterDraw documents (top left and bottom right) open on the screen. Both InterDraw documents contain scanned images cropped, copied, and pasted from InterPix documents.

Likewise, to follow a link, a user exploring a linked set of documents can select a marker icon in any type of document and choose the Follow command from a menu. As a short cut, a user can double-click on a marker icon to initiate the follow, just as he or she might double-click on an icon in a folder to open a document. Since following a link usually entails opening a document, we anticipated that users would expect to be able to follow a link by double-clicking on the marker icon.

Unlike some other hypertext or hypermedia systems that only allow links to entire documents,[1] Intermedia allows users to create bidirectional links from a specific location in one document to a specific location in another document. These "anchor points" in the documents are called *blocks*. One of our design goals, in designing the Intermedia linking functionality, was to allow the user to designate any selection region as a block that might stand alone or serve as an anchor for one or more links. The size of a block, therefore, may range from an entire document to an insertion point, depending on the selection region a user identifies as the block's extent.

For example, in an InterText document, a block might consist of an insertion point, a single character, a word, or two paragraphs. Small marker icons placed near the source and destination blocks indicate the existence of a link. As a user edits a document, the blocks "stick" to the selection they are associated with, preserving the context of the connection. If a document containing links is deleted, the links to that document are also deleted; however, the block markers at the other ends of the links remain intact, reminding users of the location of link anchors.

To help manage a large corpus of linked documents, links and blocks are assigned descriptive properties. Some of these, like user I.D. and creation time, are assigned automatically, while other properties are user-defined. Users access and edit link and block property information through property sheet dialog boxes. These dialog boxes allow users to enter a one-line "explainer," similar to the subject field in an electronic mail message. Link explainers are particularly important from a reader's perspective. If more than one link emanates from a single block, users choose the path they wish to follow from

(Continued on p. 90)

Sample session

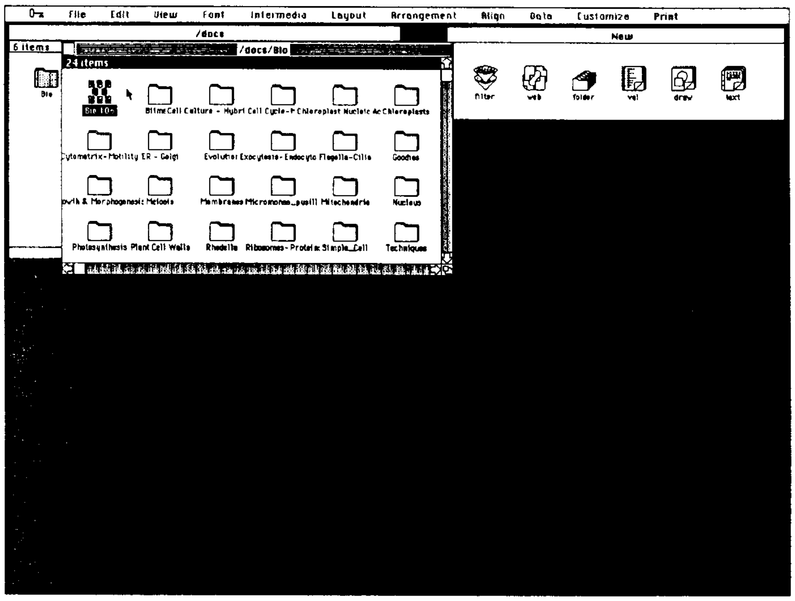
To illustrate Intermedia's user-interface features and linking functionality, this sidebar will take you on a system walkthrough designed to simulate the interaction that takes place during a hands-on Intermedia session. The screen illustrations should help you visualize the system, while the text should supply the action.

The example is taken from *Bio 106: Cell Biology in Context*, designed by Brown University biology professor Peter Heywood. Students in Heywood's plant cell biology course use Intermedia's editors, utilities, and linking functionality to write term papers and explore materials about the cell and its processes.

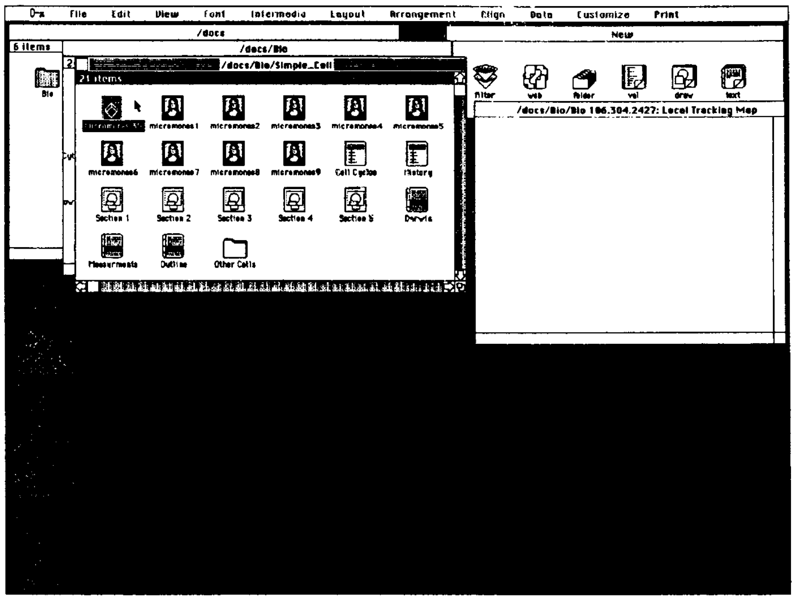
The scenario will take you through a sample session from the perspective of the biology professor in the midst of creating course materials.

Screen 1. As you can see, the Intermedia desktop includes a window manager, a graphical folder system, a menu bar, and a mouse interface. The contents of the folders reflect the underlying hierarchical structure of the file system.

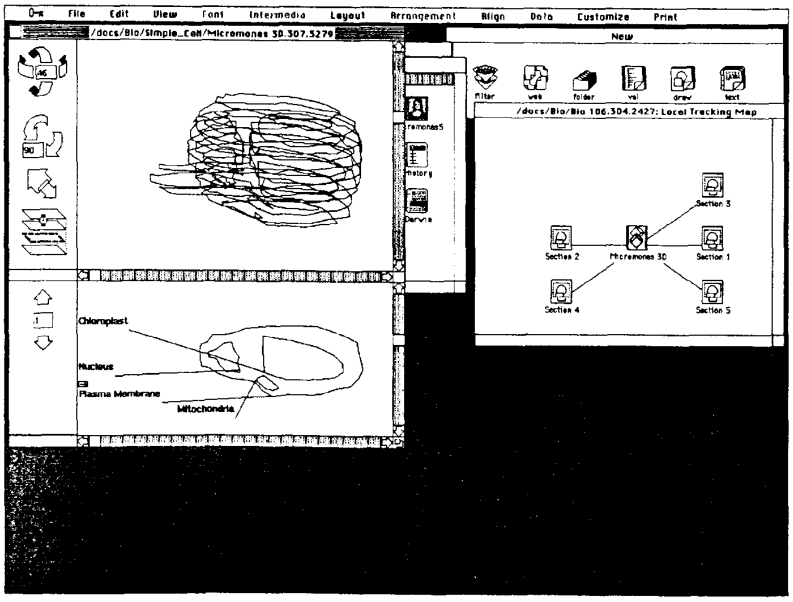
Unlike the Macintosh, Intermedia does not store application icons in the same folders with documents. Instead, application icons are stored with several other special-purpose tools in an application, or New, window that you can see in the upper right corner of the screen. The reason for this is twofold. First, users do not have to search through folders to find the applications. Even if the New window is hidden from view by overlapping windows, selecting the New command from the File menu will reveal it. Second, in a networked environment, it is best to have a single set of applications in an agreed-upon place that can be maintained and updated by a system administrator.



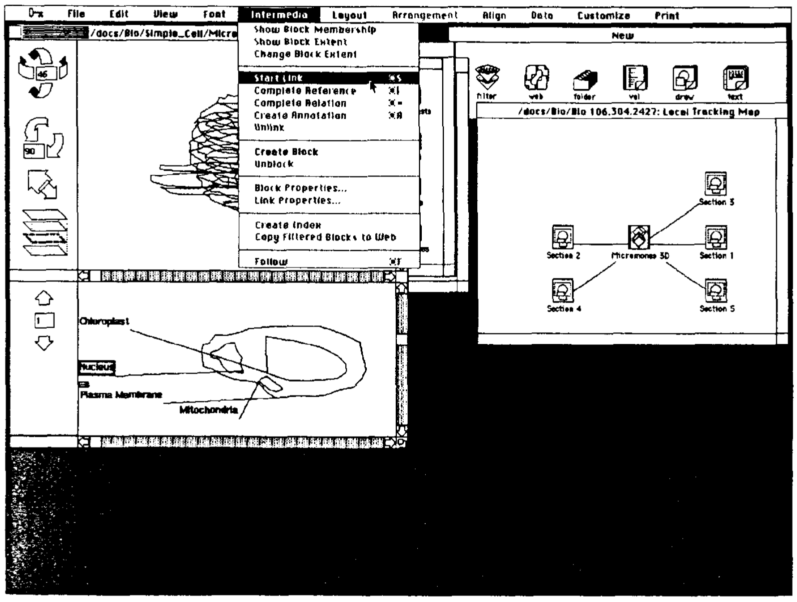
Screen 1



Screen 2


File Edit View Font Intermedia Layout Arrangement Align Data Customize Print
/docs/Bio/Simple_Cell/Micromonas 3D.307.5279
New
filter web folder val draw text
/docs/Bio/Bio 106.304.2427: Local Tracking Map
Section 3
Section 2
Micromonas 3D
Section 1
Section 4
Section 5
Chloroplast
Nucleus
Plasma Membrane
Mitochondria

Screen 3



Screen 4

Before we browse through the documents contained in the folders, follow links, or create them, we must first define a context by opening an existing web or by creating a new one. If a web is not open, we can still open and edit the documents even though no link and block information will be visible. Rather than beginning a new web, we select the icon titled "Bio 106" and choose the Open command (not pictured) from the File menu.

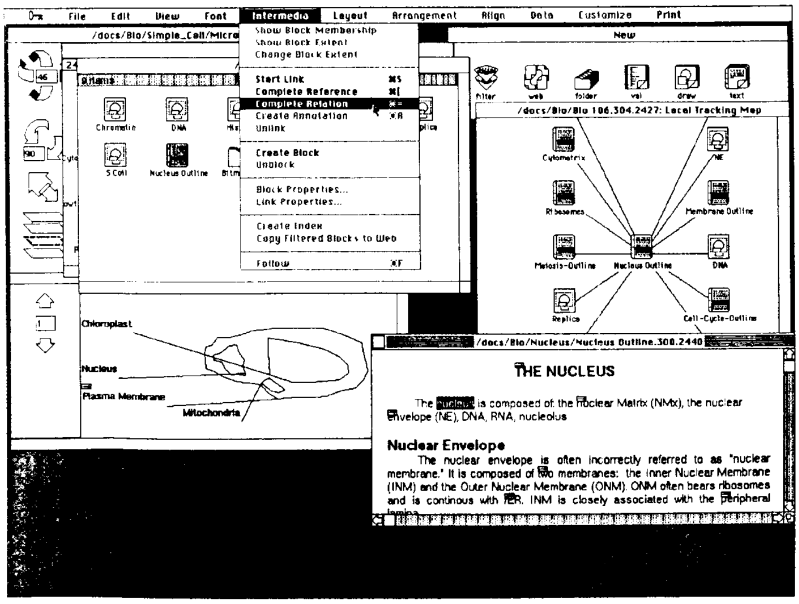
Screen 2. After opening the web, indicated by an empty local tracking map window (described below), we open a folder contained in the "Bio" folder called "Simple Cell." The icons in this cell folder represent a folder plus a number of different types of documents (one InterSpect, nine InterPix, two InterVal, five InterDraw, and three InterText). Any of these document icons can be selected, opened, and edited. We select and double-click on the InterSpect document called "Micromonas 3D" to open it.

Screen 3. When the InterSpect document opens, the application displays the entire Micromonas cell in a three-dimensional view (above left) and a single section of the cell in a two-dimensional view (below left). Students can use the tools in the palette to rotate the 3D reconstruction, to highlight the location of the 2D section currently displayed in the bottom view, and to scroll through all the 2D sections associated with the cell. Menu commands allow you to selectively hide and display different components of the cell and/or the labels.

The local tracking map, empty in the previous screen, now shows the currently active document and the links that emanate from it. Local maps are analogous to detailed street maps. They show you your current location and what location you can travel to the immediate vicinity. As you change locations, you require a new local map as a guide. In Intermedia, when the user activates a different document, either by following a link or by opening one from a folder, the local map updates or tracks the user's progress to display the new current document and its direct predecessor and successor links.

Screen 4. When we last worked on the Micromonas 3D document, we created a number of links connecting the plasma membrane to five different InterDraw documents, each containing a scanned photograph of one of the sections of the Micromonas cell used as data for the 3D reconstruction. Before we connect the plasma membrane to the remaining photographs, we decide to connect the nucleus to general information about nuclei. The first step in creating a link involves defining a block to serve as the anchor for the link. We select the label "Nucleus" as the source block of the new link (the selection is indicated by a rectangular box) and choose the Start Link command from the menu. While a link is pending, we can perform any number of other actions unrelated to link-making. Like the Copy operation common to all Macintosh-like applications, the Start Link operation is completely modeless.

Screen 5. Before completing the pending link, we browse through the folders and locate and open an already existing InterText document called "Nucleus Outline." Once the text is displayed, we select the word "nucleus" in the first sentence of the document to serve as the destination block of the link and choose Complete Relation from the menu. You will notice two different complete commands in the menu. These are similar in function, but each creates a different type of link. The Complete Relation command that we



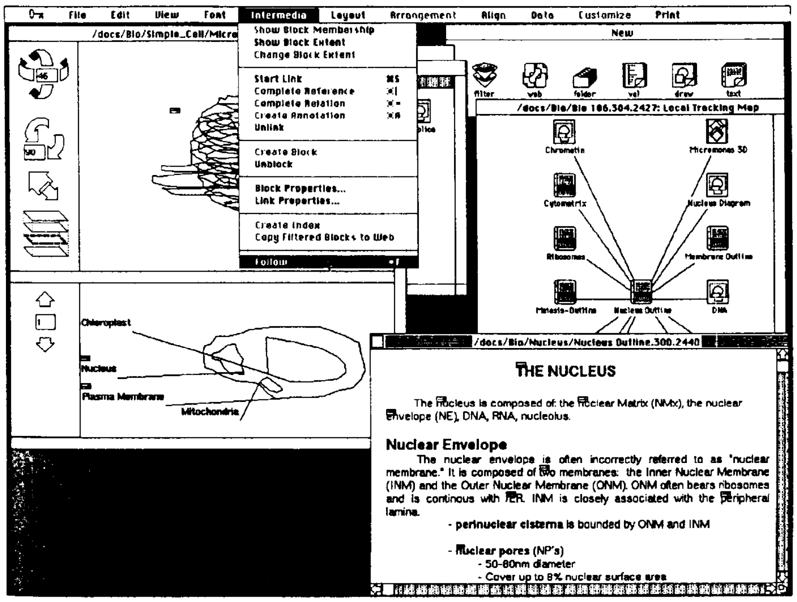
THE NUCLEUS

The nucleus is composed of the nuclear Matrix (NMx), the nuclear envelope (NE), DNA, RNA, nucleolus

Nuclear Envelope

The nuclear envelope is often incorrectly referred to as "nuclear membrane." It is composed of two membranes: the Inner Nuclear Membrane (INM) and the Outer Nuclear Membrane (ONM). ONM often bears ribosomes and is continous with ER. INM is closely associated with the peripheral lamina.

Screen 5



THE NUCLEUS

The nucleus is composed of the nuclear Matrix (NMx), the nuclear envelope (NE), DNA, RNA, nucleolus.

Nuclear Envelope

The nuclear envelope is often incorrectly referred to as "nuclear membrane." It is composed of two membranes: the Inner Nuclear Membrane (INM) and the Outer Nuclear Membrane (ONM). ONM often bears ribosomes and is continous with ER. INM is closely associated with the peripheral lamina.

- **perinuclear cisterna** is bounded by ONM and INM
- **nuclear pores** (NP's)
  - 50-80nm diameter
  - Cover up to 8% nuclear surface area

Screen 6

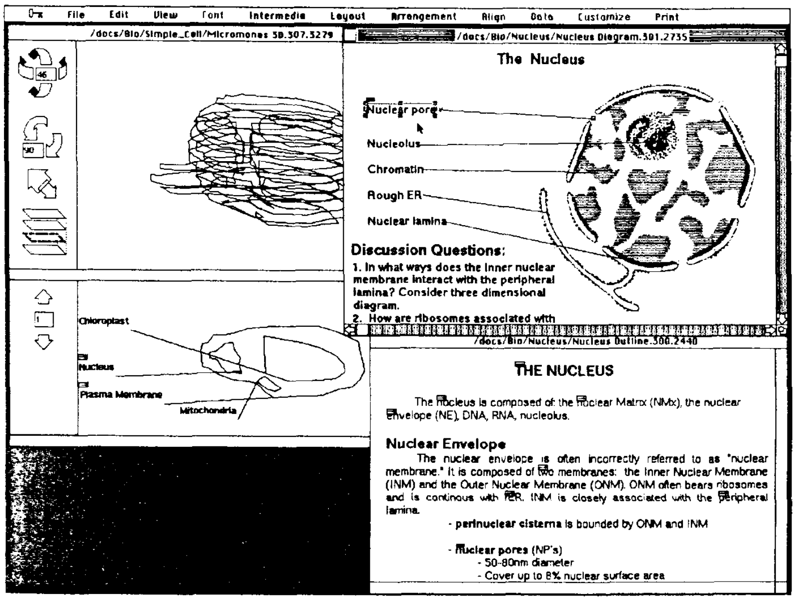
chose indicates a primary path, whereas the Complete Reference command signifies a secondary path, in much the same way as a footnote or a "see also. . ." reference. Notice that the local tracking map has been updated to show the links that emanate from "Nucleus Outline," since it is the currently active document.

Screen 6. Once the link is established, both ends are indicated with markers (arrows enclosed in rectangular boxes) and the new link is added to the local tracking map. To find other relevant material to connect to the nucleus in the InterSpect document, we enlarge the InterText window and read through the text. Since pores are important when studying simple cells such as the Micromonas cell, we select the link marker above the words "nuclear pores" and choose the Follow command from the menu to traverse the link.

Screen 7. Following the link causes an InterDraw document containing a diagram of the nucleus to open. Notice that when a link is traversed, Intermedia automatically highlights the extent of the block at the other end of the link, indicating a particular scope of information to the reader. In this case, our attention is drawn to the label "Nuclear pore" and its associated label line.

The illustration on the screen was entered into the system using a scanner. The bit map was then displayed by the InterPix application, cropped, and pasted into this InterDraw document, and the text and lines were added to complete the diagram.

Before we continue making new links, we decide to change the default "viewing specification" for link creation to "verbose" (not pictured). With the verbose option, Intermedia automatically presents a property sheet for each new link as it is created.



The Nucleus

Discussion Questions:
1. In what ways does the inner nuclear membrane interact with the peripheral lamina? Consider three dimensional diagram.
2. How are ribosomes associated with

THE NUCLEUS

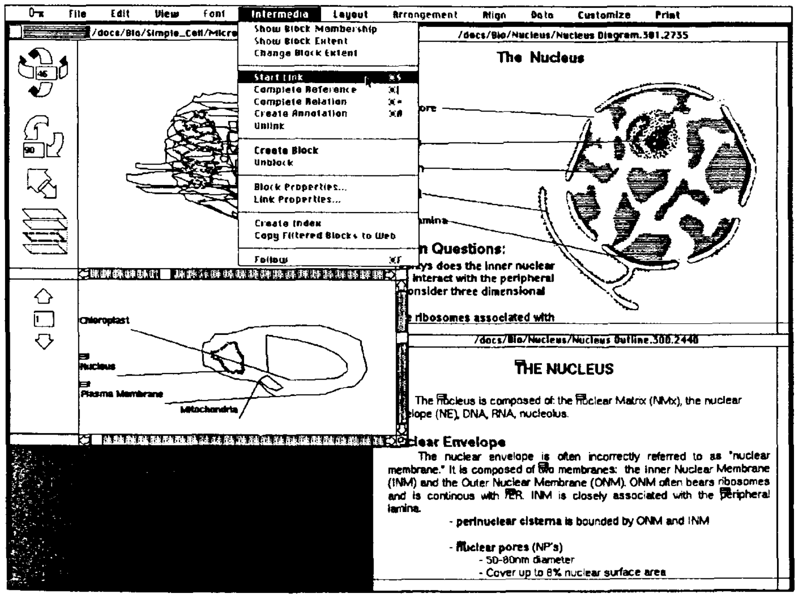
The nucleus is composed of the nuclear Matrix (NMx), the nuclear envelope (NE), DNA, RNA, nucleolus.

Nuclear Envelope

The nuclear envelope is often incorrectly referred to as "nuclear membrane." It is composed of two membranes: the Inner Nuclear Membrane (INM) and the Outer Nuclear Membrane (ONM). ONM often bears ribosomes and is continous with RER. INM is closely associated with the peripheral lamina.

- **perinuclear cisterna** is bounded by ONM and INM
- **nuclear pores** (NP's)
  - 50-80nm diameter
  - Cover up to 8% nuclear surface area

Screen 7 (The illustration of the nucleus was originally published in *Molecular Biology of the Cell* by Bruce Alberts et al. and is reprinted with the permission of Garland Publishing, Inc.)



THE NUCLEUS

The nucleus is composed of the nuclear Matrix (NMx), the nuclear lope (NE), DNA, RNA, nucleolus.

lear Envelope

The nuclear envelope is often incorrectly referred to as "nuclear membrane." It is composed of two membranes: the Inner Nuclear Membrane (INM) and the Outer Nuclear Membrane (ONM). ONM often bears ribosomes and is continous with RER. INM is closely associated with the peripheral lamina.

- **perinuclear cisterna** is bounded by ONM and INM
- **nuclear pores** (NP's)
  - 50-80nm diameter
  - Cover up to 8% nuclear surface area

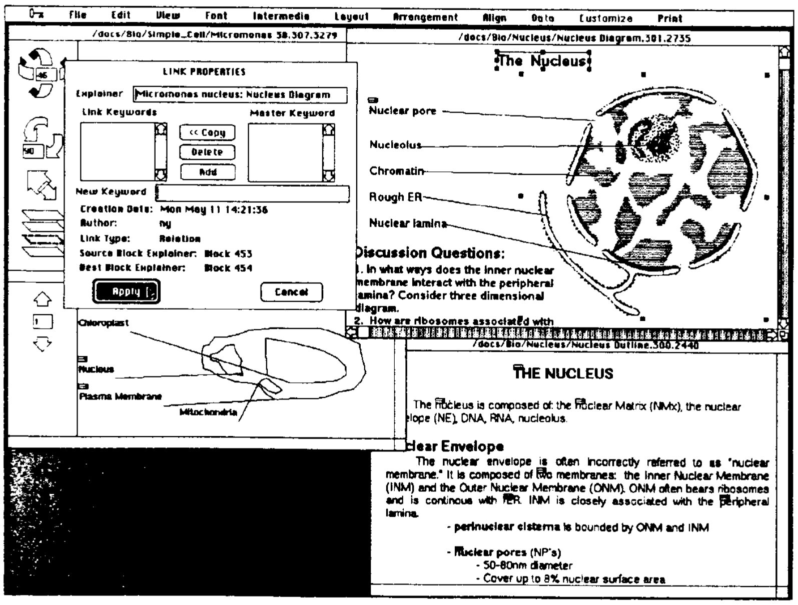
Screen 8

Screen 8. We activate the Inter-Spect document by clicking once in the window and select the nucleus. This time we decide to select the component itself rather than the label. When students follow the link from the InterDraw diagram to the three-dimensional representation, their attention will be drawn to the nucleus (the source block) in both the 2D and 3D views. As in Screen 4, we select the Start Link command to initiate a new link.

Screen 9. Next, we reactivate the "Nucleus Diagram," select the title of the diagram and the scanned illustration as the destination block for the link, and choose Complete Relation from the menu (not pictured). After we issue the complete command, a link property dialog box appears, allowing us to fill in descriptive information about the link. We replace the default text, "Link 35," with the more meaningful explainer shown in Screen 9.

Screen 10. Now we will skip ahead a few steps. After creating the link from the nucleus in "Micromonas 3D" to the InterDraw diagram, we reactivated the Inter-Spect document and used the bottom tool in the palette to scroll to the next two-dimensional section. Since the label "Plasma Membrane" has already been defined as a block for another link, we decided to select the existing marker as the source point for our new link.

Before we are ready to complete the link, we have to create a new document. We return to the "Simple Cell" folder, open an InterPix document (bottom left) containing a photograph that corresponds to the section currently displayed in the InterSpect document window, and crop and copy a portion of the photograph into



LINK PROPERTIES

Explainer: Micromonas nucleus: Nucleus Diagram

Link Keywords

Master Keyword

New Keyword

Creation Date: Mon May 11 14:21:36

Author: ny

Link Type: Relation

Source Block Explainer: Block 453

Dest Block Explainer: Block 454

Discussion Questions:

1. In what ways does the inner nuclear membrane interact with the peripheral lamina? Consider three dimensional diagram.

2. How are ribosomes associated with

THE NUCLEUS

The nucleus is composed of: the nuclear Matrix (NMx), the nuclear elope (NE), DNA, RNA, nucleolus.

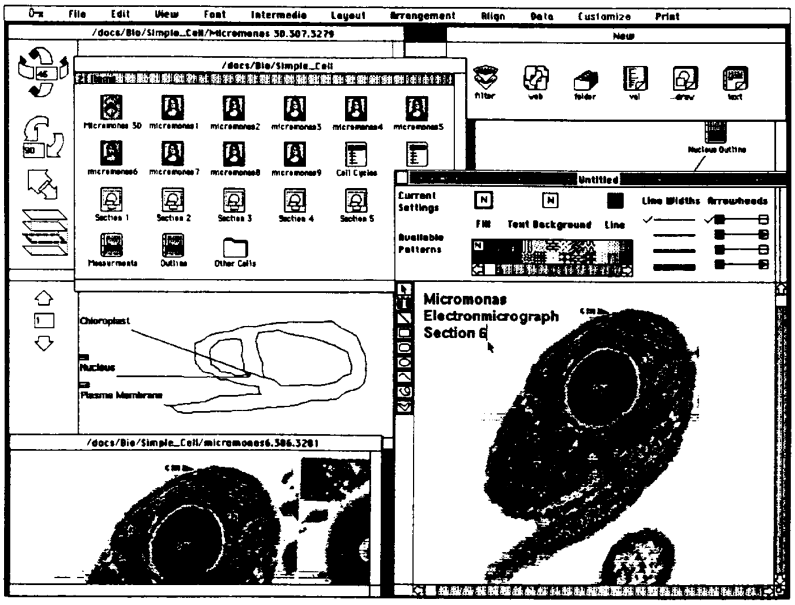
clear Envelope

The nuclear envelope is often incorrectly referred to as "nuclear membrane." It is composed of two membranes: the Inner Nuclear Membrane (INM) and the Outer Nuclear Membrane (ONM). ONM often bears ribosomes and is continous with RER. INM is closely associated with the peripheral lamina.

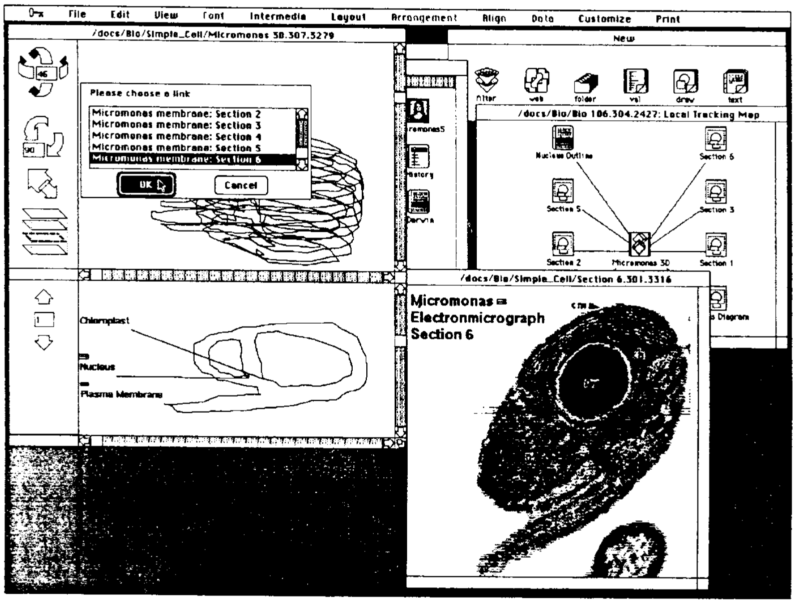
- perinuclear cisterna is bounded by ONM and INM

- nuclear pores (NP's)
  - 50-80nm diameter
  - Cover up to 8% nuclear surface area

Screen 9



Screen 10 (The electronmicrograph of Micromonas was originally published in *The Journal of Phycology* and is reprinted with the permission of the editor.)

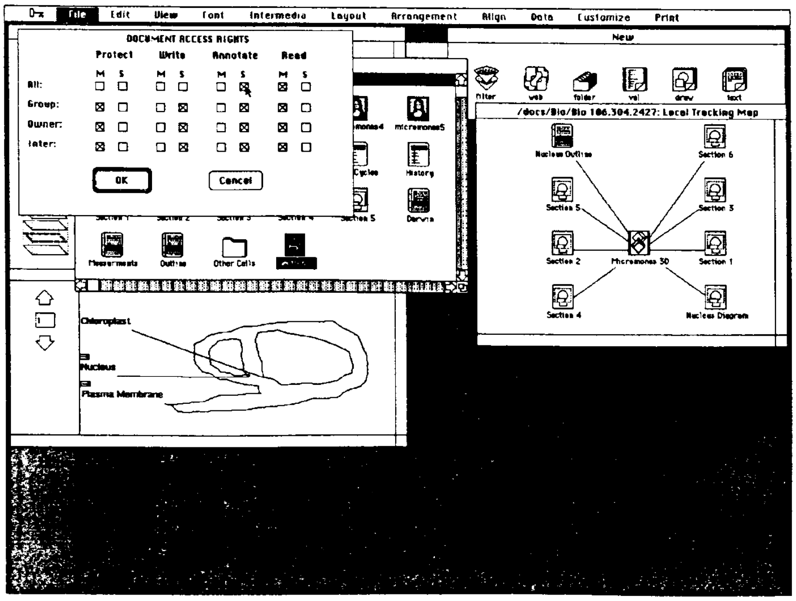


Please choose a link

Micromonas membrane: Section 2
Micromonas membrane: Section 3
Micromonas membrane: Section 4
Micromonas membrane: Section 5
Micromonas membrane: Section 6

Micromonas Electronmicrograph Section 6

Screen 11



DOCUMENT ACCESS RIGHTS

	Protect		Write		Annotate		Read	
	M	S	M	S	M	S	M	S
All:	☐	☐	☐	☐	☐	☒	☒	☐
Group:	☒	☐	☐	☒	☐	☒	☒	☐
Owner:	☒	☐	☐	☒	☐	☒	☒	☐
Iater:	☒	☐	☐	☒	☐	☒	☒	☐

Screen 12

the clipboard. We paste this image into a new InterDraw document, created by double-clicking on the draw icon in the New window. Finally we add some text to accompany the photograph (bottom right).

Screen 11. Here, we have completed editing the new InterDraw document and have hidden the palettes to unclutter the screen. We have also completed the pending link, using the text "Micromonas Electronmicrograph Section 6" as the destination block of the link. After the link was established, we double-clicked on the marker associated with "Plasma Membrane" in the InterSpect document to traverse the new link. Since more than one link is associated with the selected block, Intermedia presents a dialog box containing the explainers for each link. We select the link we just created and click on "OK." Since the document at the other end of the link is already open on the screen, following the link will simply activate the document and highlight the extent of the destination block (not pictured).

Screen 12. Before ending our session, we save and close the new InterDraw document, select its icon, and choose the Access Rights command from the menu. The dialog that appears allows us to add or subtract access rights for different groups of users. For this document, we decide to add Annotate rights for all users of the system. This means that any user may create links to or from the document but may not edit its content. Before exiting the system, we save and close the open InterSpect document and the Bio 106 web.

(Continued from p. 83)

a list of link explainers presented in a dialog box.

Property sheets also allow users to add keywords. Although still under development, these keywords, along with the default information assigned to links and blocks, will provide users with a mechanism for searching the document corpus. A keyword search will yield a list of explainers associated with all the blocks or links meeting the search criteria. Each item in the resultant list will be automatically linked to its corresponding block or, in the case of links, to the corresponding source block of each link. For example, a student could search for all links containing the keywords "Pope" and "Heroic Couplet" that were created by the professor after a certain date.

Link and block properties help manage complexity within the Intermedia environment, but the notion of context is even more crucial. In some systems, links are global—all links are available at all times to all users. In such systems, links become an integral part of the documents. In Intermedia, block and link information is not stored within individual documents but is superimposed on them. *Webs* maintain the block and link information, allowing one or more users to work within their own context undistracted by blocks and links created by others sharing the same computing resources. Most importantly, users do not see hypermedia as an alternative to their desktop environment; rather, they see it as an integral technique tying together documents in that environment.

Currently, opening a web imposes a particular set of blocks and links on a set of documents while that web is open. Thus, webs allow different users to impose their own links on the same document set. Although only one context can be viewed at a time, users can easily switch contexts by closing one web and opening another. In the future, webs will also serve as the focus for keyword searching operations.

Intermedia differs from most other hypermedia systems in that it allows multiple users to both follow and create links concurrently in the same web. Intermedia incorporates a system of user access rights that helps manage multiple users sharing large bodies of connected material. Due to the hypermedia functionality of Intermedia, the access rights scheme builds on the protection mechanisms offered in most file systems where users either have "read" permission or "write" permission to files and directories. Intermedia adds "annotation" permission to the other two forms of access rights. This allows users to add links to a document that they are not allowed to edit.

Intermedia: the construction

Intermedia not only provides a rich environment for authors and browsers but also for developers, furnishing a set of tools that facilitate the creation of new applications adhering to the Intermedia paradigms.

In designing the Intermedia system, we believed that consistency among applications was crucial, since the system encourages quick transitions from one application to another. User-interface consistency is not always easy to achieve, however.[5] In part, this may result from carelessness. But, more often, interface inconsistencies result from not quite identical implementations of features already implemented elsewhere in a system. The Apple Macintosh represents a prime example. The system has clearly defined user-interface paradigms, and new programs almost always use a number of the same functions that exist in hundreds of other Macintosh programs. Even so, software developers must reimplement most of the "standards" (selection, resizing, dragging, etc.) from scratch because the Macintosh Toolbox provides the mechanisms for building them but not the implementations. Often these programmers miss an important feature or user-interface detail that users immediately notice.

To build Intermedia, we needed a development environment that would help programmers create a multiuser system with consistent, direct-manipulation applications, plus the ability to link together the contents of documents created with those applications. Faced with the task of developing a relatively large, interactive system in an ambitiously short timeframe, we needed a set of development tools that would help us

- remove the burden of user-interface consistency from the application programmer
- adopt an existing user-interface standard
- allow small groups of programmers to work on different parts of the system in parallel
- facilitate the integration of modules developed by different groups
- avoid as much duplication of effort as possible, and
- create a system that would be extensible and suitable for prototyping new applications.

We created such an environment by building some of the pieces ourselves and adapting and integrating tools developed by others. This resulted in a layered set of tools that allows programmers to develop applications conforming to the user-interface standards.

In particular, we started with CadMac, Cadmus Computer's implementation of the Macintosh toolbox under the 4.2 BSD Unix operating system; supplemented it with an object-oriented programming language called Inheritance C (developed at Bolt Beranek and Newman); and added a C version of Apple's MacApp—a set of classes for creating "generic" Macintosh-like applications. On this, we superimposed several crucial building blocks from which any number of end-user applications and utilities can be constructed. *The Proceedings of the 1986 Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*[6] provides a detailed technical description of the architecture of the development environment that we summarize below.

Object-oriented programming. The technique of object-oriented programming has gained a great deal of recognition as a superior approach to programming tasks. Studies have shown significant reductions in both development time and size of source code when such techniques were used,[7] with a significant increase in the amount of reusable code. One criticism of programs written using object-oriented programming techniques is that they tend to be slower than comparable conventional programs. First, much of this view relates to historical speed problems in early versions of Smalltalk where object-oriented code was interpreted rather than compiled. With the advent of object-oriented compilers and optimization techniques, speed is no longer an insurmountable problem. Second, if you use an appropriate object-oriented system,[8] you can carefully write and optimize reusable chunks of code.

We selected an object-oriented programming language for the Intermedia project partly because of the reduction in development time it promised, but mostly for the benefits it affords to a group-development effort. A team of developers can agree on a shared set of parent classes and their corresponding abstract methods.

The use of abstract methods—or templates—clarifies the behaviors an application programmer must implement. Then, working individually, the developers can create appropriate subclasses—defining one class of objects in terms of other classes—that will respond to a single set of messages and that can be easily integrated into one program. When a programming task is divided, one member of the team can implement the code that will coordinate the sending of messages to objects while the other members of the team work on defining and refining the object classes.

The applications framework. While an object-oriented programming language provides a number of features that facilitate team efforts, supplementing those features with a set of classes that define common behaviors—an applications framework—insures a greater degree of user-interface consistency in the system under development.

Our choice, Apple's MacApp, represents such a companion to an object-oriented programming language and provides a framework for constructing Macintosh applications.[9,10] MacApp defines classes with a combination of abstract and nonabstract methods that encapsulate the behavior of the Macintosh user interface. A tiny program (on the order of 10 to 20 lines) bound with MacApp suffices to create a skeletal Macintosh-like application with menus; blank windows that can be moved, resized, and scrolled; data that can be stored and

Review of object-oriented programming principles

Not surprisingly, the fundamental notion in object-oriented programming is that of objects. An object-oriented program is a system of interacting objects. Objects encapsulate data and the algorithms that specify the behavior of that data. Operations on an object can take place only through a well-defined interface to the object's behavior; the actual implementation of the behavior is hidden from everyone but the designer.

The data structure components of an object are known as its *fields* or *slots*. The routines that can act upon an object of a particular type are called *methods*. These methods are the primary means through which the fields within an object may be manipulated or modified. Other objects invoke an object's method by sending a message to the object. Then, the object interprets the message and the appropriate method is performed.

Classes (also known as object types) are templates defined by programmers that describe the properties and behaviors of a set of common objects. An object is actually an instance of a class template, typically created as a program is running. Each object is a copy of the class template. Thus, each object has the same number and types of fields, and only differs from other objects in the class in the data in those fields. All objects in a class share the same methods, typically by pointing to a common method table or dictionary. While class templates provide a basis for modularity, subclassing—the ability to define one class of objects in terms of other classes—is one of the object-oriented programming concepts from which much leverage is gained.

Subclasses inherit the characteristics of higher-level ancestor classes. An object in a subclass contains all the same field types and methods as an object in the parent class. In defining a subclass, a programmer can add fields and methods or redefine methods that one of its superclasses originally implemented. A redefined method can implement a behavior completely different from the original method, or it can merely slightly modify or extend the behavior of its parent.

Refining, or overriding, a method of a class makes it possible to considerably reduce the amount of code that an application programmer needs to write; the only code necessary is that which explains how a method differs from the parent method. The programmer is guaranteed that the methods he or she did not override will respond properly to any messages sent to the object. This process of redefining and extending a class of objects in terms of another class is important because it enables programmers to use and modify existing parts of a system without having to understand the details of their implementation.

For example, say we define a class called Rectangle. All the objects that are instances of this class will have two Point fields (the top-left and the bottom-right corners of the rectangle). The class will have methods for calculating area and drawing the rectangle. If we wish to draw a rectangle on the screen, we first create a Rectangle object and then send a message to invoke the Draw method. To create a more specialized object that draws a rectangle and prints text inside the rectangle, we would define a subclass of Rectangle, called TextRect. No existing code has to be rewritten. Instead, in the definition of the TextRect class we can add a character string field, override the Draw method inherited from the parent class, refine its behavior to do what its parent did, and draw the text. Like Rectangle objects, TextRect objects will respond to FindArea messages, even though we did not add any code for calculating area in the TextRect class.

In the above example, the Rectangle class served as a template for the subclass TextRect. However, it often helps to define less specific templates than the Rectangle class. For instance, a parent class of Rectangle called Shape might have been created with two abstract methods, Draw and FindArea. An abstract method contains no code; it exists only for the purpose of being overridden. To create a new shape, a programmer would subclass Shape, add appropriate fields, and override the Draw and FindArea methods.

Likewise, a program that displays many different shapes on the screen might contain a list defined to point to objects of class Shape or any subclass of Shape. Each object in the list inherits the Draw method from the Shape superclass, but has overridden it with code appropriate for drawing the specific object. Since all shapes are guaranteed to understand the same message protocol, we can display a whole screenful of shapes by merely sending the same draw message to each object in the list without knowing exactly what type of shape objects are in the list.

Objects descending from the same parent class are essentially "plug compatible;" each understands the same messages as the others, yet each performs the task in its own way. This modularity allows the transparent creation and insertion of new subclasses into the program.

retrieved; and views that can be automatically laser printed. This default program, however, has a blank view. To write an application with views that render something in the windows, the programmer must subclass several base classes provided by MacApp. The most important of these classes are

- The Object class, which manages the freeing of memory and the cloning of new objects. It is the parent of all other classes.
- The Application class, which contains methods for launching an application, displaying the menu bar, managing the main event loop, and creating and initializing appropriate document objects.
- The Document class, which maintains the data model for the program. Document objects contain all the basic information for saving and restoring the data and managing several other objects—such as Window objects, Frame objects, and View objects—involved in viewing the information contained in the document.
- The Window class, which manages all the operations pertaining to windows, including opening, closing, resizing, moving, activating, and redrawing.
- The View class, which manages the rendering of the data contained in the document and passes on mouse events to the appropriate objects within the view.
- The Command class, which is the template from which command objects are generated to respond to outside actions from the menu, the mouse, or the keyboard. Since command objects can be maintained on a stack, multilevel undo and redo are easily implemented.

By subclassing these and other MacApp classes, a programmer builds a model for the data in an application, creates the windows and frames in which the information will be viewed, and describes how the user can interact with that information.

The Application class has perhaps the greatest impact on the developer. This class contains the methods necessary for an application's most basic behavior. For example, it includes methods for launching an application, running the main event loop, dispatching events to the appropriate event handler, and creating, closing, and deleting documents. In the case of a user selecting a command from a menu, the Application object interprets the mouse press and sends a message to the currently selected object's DoMenuCommand method. A programmer does not have to consider flow-of-control issues since an Application object handles all user-initiated events, such as mouse presses, keystrokes, and menu selections, and dispatches those events to the appropriate target object.

Two building blocks were initially implemented. Later, it was discovered a third building block was needed.

As an applications framework, MacApp promotes consistency in a multi-application environment by eliminating the need for programmers to reimplement any of the user-interface features required for the shell of a Macintosh-like application. The framework insures that each application will have windows, menus, dialog boxes, and other basic components that look and behave the same way as all others in the system.

Building blocks. While object-oriented programming provides the structure and methodology for cooperative development, and MacApp provides a set of base classes from which to build an application, these two components alone are not enough to create a fully functional development environment for a group of cooperative developers.

Still missing is a component that helps developers render and manipulate the data for their particular application. To this end, we have implemented several *building blocks*—sets of reusable classes that implement basic functions common to multiple applications.

The philosophy behind the building blocks is that they should encapsulate important end-user functionality—both input and output components—and provide both a user interface and a programmer interface. Instances of these building blocks can be incorporated directly into an application; the application programmer can use part or all of a building block's functionality as it exists or modify the functionality to suit a specific application. To support the development of applications within Intermedia, we initially implemented two building blocks—a Text Building Block and a Graphics Building Block—and later found the need for a third—a Table Building Block.

The Text Building Block permits the inclusion of text anywhere within an application. It makes it possible to provide exactly the same interface for displaying, editing, and formatting multifont text throughout the system. An entire application, such as a text editor, or some part of an application, such as the input field of a palette, can be based on the Text Building Block.

Just as the Text Building Block allows the inclusion of text anywhere within an application, the Table Building Block facilitates the incorporation of tabular data. A programmer can use the Table Building Block as the backbone of a spreadsheet program or a database interface, or to integrate one or more tables into any other type of application. For example, Release 3.0 of Intermedia will include a videodisc application with tables for storing data such as frame numbers, sequence names, and playing times of video images.

The Graphics Building Block (GBB) lets programmers incorporate graphics, such as lines, rectangles, circles, icons, and polygons, into their applications. This building block defines a number of shape classes with methods for drawing, highlighting, selecting, resizing, and moving. The GBB also subclasses MacApp's View class so that the subclassed graphics GView contains a list of all objects to be rendered on the screen. To illustrate how building blocks are used in general, we will focus on the GBB.

Programmers can take advantage of a building block such as the GBB in one of four ways. First, a programmer can use the building block functionality in its entirety. For example, to create a structured graphics editor similar to Apple's MacDraw in which users can draw a variety of different shapes on the screen, rearrange them, group them, and perform various other editing operations, a programmer could use most of the GBB's shape classes, subclassing where necessary, and then add application-specific user-interface features such as alignment, tool palettes, and style palettes.

In the second case, a programmer can eliminate functions inherited from a building block. For example, one method associated with polygon objects in the GBB allows users to move polygons by clicking on them and dragging. In InterSpect, data files—not users—govern the placement of each polygon relative to other polygons in a three-dimensional object, so the developers override GPolygon's Move method to eliminate the dragging functionality. In all other respects, polygons behave in InterSpect as defined in the GBB.

Third, you can override methods to add increased functionality to building block classes. The use of icons in Intermedia's desktop application illustrates the addition of functionality to a building block's method. The desktop application subclasses GBB icons—simple bit maps—and overrides the Draw method so that a text string, representing a document's name, always appears below the icon (see Screen 1 in the sidebar, "Sample session").

In the fourth case, a programmer can override methods to change the behavior of building block classes. An example in InterSpect clearly illustrates this. To indicate that an object has been selected, the GBB uses "handles" to indicate highlighting. In InterSpect, however, the GBB's GSelection method is overridden to substitute bold outlines as a highlighting method.

With these four options available, application developers have enough flexibility to create innovative interfaces, but are not burdened with the implementation of functions that should behave identically across applications. The building blocks complement MacApp by providing a means of achieving internal as well as external consistency among applications.

Adding shared functionality. By using the tools described above—an object-oriented programming language, MacApp, and building blocks—a programmer could create applications that adhere to the Macintosh-style user-interface paradigms. In our requirements for Intermedia, however, we identified the need to run multiple applications on the desktop as well as the need to link the contents of documents together. These two requirements made it necessary for us to extend, and in some cases alter, the existing Macintosh user-interface paradigms. To this end, we kept MacApp as the first layer of our system and then subclassed most of the MacApp classes to create an Intermedia layer. The way the Intermedia layer extends the functionality of MacApp illustrates the ease with which features shared by many applications can be implemented using our object-oriented development base.

Running multiple desktop applications and linking document contents were identified as requirements.

Briefly, the type of additional functionality the Intermedia layer supports includes the creation of links between a selection in a source document and a selection in a destination document. To attain the desired consistency, the Intermedia layer subclasses MacApp's Document, View, and Application classes. In MacApp, the Document class manages the reading and writing of an application's data model while the View class manages the rendering of that model. IntDoc and IntView, the Intermedia layer's document and view classes, extend the MacApp functionality to include the reading and writing of link information to a relational database and the rendering of the links. The Intermedia layer's application class, IntAppl, adds the functionality necessary to interface with Intermedia's folder system.

Like MacApp, the Intermedia application framework "calls" the applications through the use of abstract methods. By defining an abstract method, the framework indicates to the application developers that it is the responsibility of a building block or an application to provide a concrete implementation of that method. For example, IntView provides abstract methods for displaying block markers concretely implemented by the various building blocks.

The linking functionality is implemented largely in the Block class that exists at the Intermedia layer and inherits from MacApp's Object class. Since there must be a block at either end of a link, blocks are created each time a link is made, unless the user attaches one end of the link to an already existing block. A Block object keeps track of all links that emanate from it by pointing to a Link Array object that, in turn, points to the appropriate Link objects. Therefore, users can access a link through a block to follow the link or view its properties.

The methods of the Blocks include ones for starting links, completing links, following links, viewing properties, and several others. Certain appropriate menu items become available when a BlockMarker is selected. The menu items for showing the extent of the block, starting a link, deleting the block, and viewing the block properties are enabled if the block to which the Block object points has no links. If that Block has at least one link, all of the previously mentioned options are available, along with following, viewing of link properties, and deleting the link. All of these actions are done using Block methods that themselves may use fields as well as methods of the associated Link objects.

This portion of the linking functionality is shared in its entirety by all Intermedia applications, leaving only a few details for the developer to implement in order to integrate linking into a new application. With a non-object-oriented development base, each application programmer would have been forced to identify and call procedures from appropriate subroutine libraries for saving, restoring, creating, deleting, and viewing links, and to do all this in the appropriate order with the correct parameters. Using the Intermedia layer augmenting MacApp, the applications framework essentially alerts the programmer, who must implement only those methods that are specific to his or her application. All other functionality is shared as standard fare by all developers. Larry Rosenstein of Apple Computer has whimsically labelled this a "don't call us, we'll call you" programming methodology.

Building an application. While the Block methods in the Intermedia layer handle the core of the linking functionality, the methods of the individual application's View objects handle the way blocks are displayed in different applications. The display methods include the creation and display of marker icons, scrolling to a block after a follow, and highlighting the extent of a block. In the GBB, these methods, defined at the Intermedia layer as abstract methods of IntView, are implemented in the subclass GView. An application that, in turn, subclasses GView inherits a whole hierarchy of functionality, part of which includes the display of

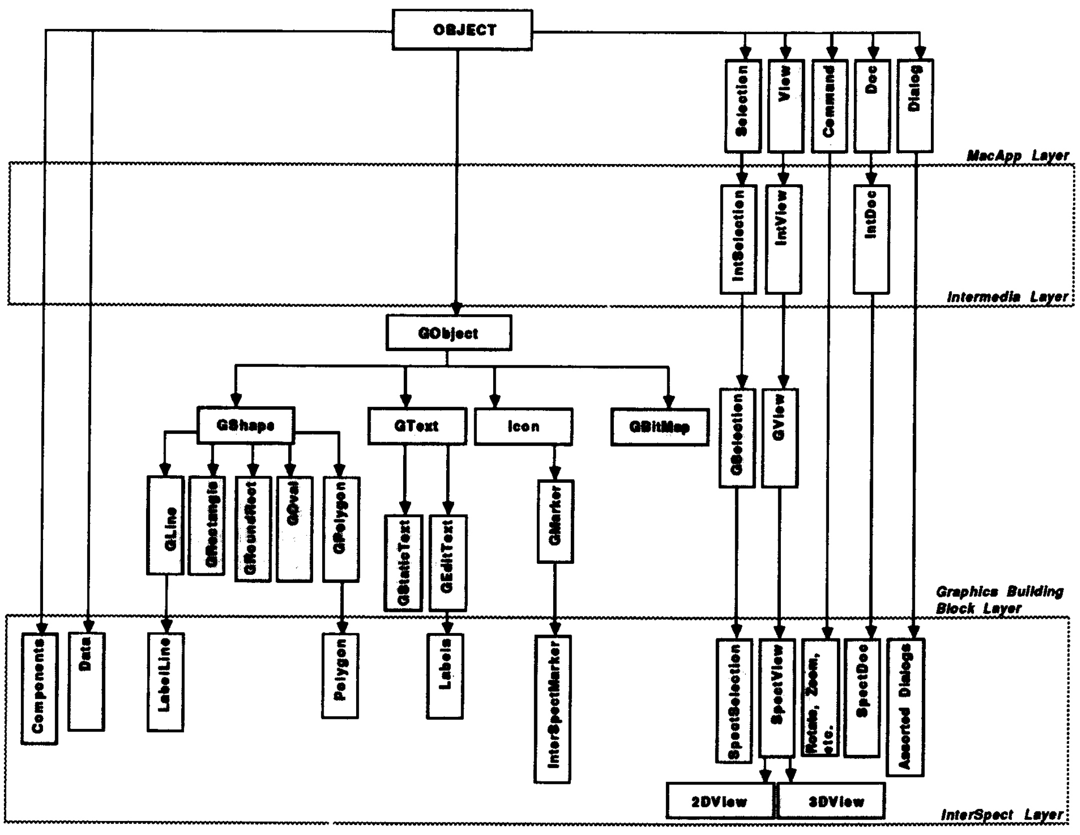


Figure 2. Inheritance hierarchy for the InterSpect unit.

blocks (see Figure 2). The application developer then subclasses GView and the associated GObject, and uses the Dialog, Control, List, and other classes to create a customized application.

Every application added to the Intermedia environment has a number of features, like zooming and rotation in InterSpect, that differ from functions implemented at the MacApp, Intermedia, or building block layers. In this case, the programmer creates new objects that are simply subclasses of MacApp's generic Command or Object classes. While the development tools do not directly simplify the implementation of these features, general object-oriented techniques help structure the thinking of programmers about their application-specific problems.

Developing in parallel. When building applications such as InterSpect in parallel with other applications that run in the same environment, we initially consider each application as a separate entity. Programmers build and debug their programs as independent applications, each taking advantage of the inheritance hierarchy provided by the development tools. The extreme modularity of the object-oriented environment makes it possible to take applications developed separately and, without requiring recompilation, integrate them into a single system.

For application integration, the Intermedia system contains a Framework application. When programmers develop an application, they compile the code for each application object separately from the code for all other objects. The compiled code minus the application object is called a *unit*. For example, InterSpect has an application object, called SpectAppl, that is a subclass of the Intermedia layer's application object, IntAppl. The SpectAppl object contains a method called DoMakeDocument exclusively for creating InterSpect document objects. To test InterSpect, the programmer binds the InterSpect unit to the SpectAppl object. When ready to integrate InterSpect into the Intermedia system, the programmer instead relies on the framework's application object (FrameworkAppl, also a subclass of IntAppl), which contains a DoMakeDocument method for creating not just InterSpect documents, but many different document types. When integrating InterSpect into the Intermedia system, the programmer adds InterSpect to the list of document types specified in FrameworkAppl's DoMakeDocument method and then binds InterSpect and all other application units with the FrameworkAppl object rather than the SpectAppl object, without ever having to recompile the InterSpect unit.

The capability to reuse the same units without recompilation for independent testing and for creating an integrated system was instrumental in the success of our parallel development effort. Developers could work on their portions of the project independently, knowing that the effort of integration would be minimal as long as they worked within the structure provided by the building block and the Intermedia application framework. In particular, developers could count on inheriting all the linking functionality as soon as their application was bound and run with the FrameworkAppl object.

Measuring success

The concept. To assess the power and utility of hypermedia, IRIS is conducting a series of experiments at Brown that introduce Intermedia into existing courses and work settings. To date, Intermedia-based materials and applications have been used in a plant cell biology course and an English literature course involving about eight users who might be classified as authors and 80 students who primarily used the system as browsers (although many experimented with creating their own documents).

As evidence that users appreciated the multiple applications provided by the Intermedia framework, two substantial linked bodies of material (approximately 850 English-related documents and 200 biology-related documents) were created that included documents of all available types.

Authors and browsers alike learned to use the system with almost no training. Even though experienced Macintosh users learned more quickly than others, no user required more than a week or two to feel comfortable using Intermedia and all its available applications. The approximately 3,000 links created by the eight authors seems to indicate that link-making as well as link-following posed little or no difficulty to the users. Although there is no empirical evidence to show that consistency among applications is related to ease of use, we believe it is a strong factor. In fact, we believe the consistent application framework with seamless inclusion of linking allows users to ponder new applications, taking for granted that linking will be incorporated into those applications as a standard feature.

An examination of the course materials created by the instructors in this study indicated that each used Intermedia successfully (measured by a substantial increase in the critical thinking skills of the students[11]), despite the fact that each instructor used the system in a fundamentally different way. We believe this study points to the potential value of multiuser hypermedia systems across a wide range of academic disciplines.

The construction. With the generic MacApp layer, the Intermedia layer, and three crucial building blocks in place, we can now build and integrate new applications into the Intermedia environment in a matter of weeks. By systematically implementing abstract methods and overriding other methods found in layers above the application layer, developers can create applications guaranteed to have common functionality consistent with all other applications. While it is, of course, possible to institute inconsistencies, it takes more effort to be inconsistent than consistent.

We can directly attribute the successful and rapid development of the Intermedia environment to object-oriented programming techniques. These techniques, in combination with other factors, enabled us to attain each of our goals for the project. We were able to remove the burden of user-interface consistency from the application developer, adopt an existing user-interface definition, allow groups of programmers to work in parallel, integrate separate modules without recompilation, avoid considerable duplication of effort, and create an extensible system suitable for the rapid development of new applications. *The Proceedings of OOPSLA 86*[6] provides a more detailed description of the specific measures we used as a basis for these conclusions.

Despite the substantial benefits of our development base, as with every system, we encountered problems and drawbacks. Our most serious problem stemmed from working in an extremely layered environment. Although inheritance certainly saves programmers an enormous amount of work, it can prove quite time-consuming in an environment that does not support incremental compilation. When working with a Unix system using the C programming language and an object-oriented preprocessor, changes in one layer are not automatically propagated to other lower-level layers. In our case, when we changed fields and methods in a parent class such as IntView, we had to recompile all layers inheriting from that class. These often required 45 minutes or longer. Although we attempted to minimize the number of recompilations, they were often unavoidable during the period we worked on all layers of the system simultaneously. We did manage to structure the working environment so the recompilations would not prevent other people from working, but this scheme added the expense of keeping duplicate copies of the source code and producing new releases of the system every week. Even though we used a source code control system to facilitate release tracking, we still had to be extremely careful to include the most up-to-date versions of every layer in each release.

Through the use of abstract methods, object-oriented programming provided us with a concrete structure that could be shared by each of the applications in the system. With object-oriented programming and MacApp, we could structure the whole system in such a way that each part could be worked on independently with the guarantee that integration would be easily accomplished and common functions would behave identically in each of the applications.

As we look toward the future, we plan to expand Intermedia, both from a user's and a programmer's point of view. Development of new applications and system features are under way to provide links to and from video and audio, more complex filtering and information retrieval, better visualizations of connections between documents, and support for capturing and replaying paths through a web. On the development side, we plan to implement building blocks for handling controls such as sliders and scrollbars, for abstracting menus, for providing MIDI music recording and replay, and for providing more sophisticated text-editing features. More importantly, we hope to persuade the software development community that (1) application development will be most fruitful when that community at large embraces object-oriented building blocks and frameworks and (2) hypermedia will only be readily accessible when a common linking protocol is adhered to by all third-party software creators. □

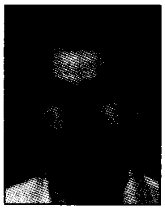
Acknowledgments

Intermedia is the culmination of two years of intense effort by a large team of developers led by Norman Meyrowitz. We would like to thank Tim Catlin, Helen DeAndrade, Page Elmore, Charlie Evett, Matt Evett, Nan Garrett, Allan Gold, Ed Grossman, Karen Smith, Tom Stambaugh, Ken Utting, Dave Bundy, Dan Stone, Steve Williams, Bill Shipp, Marty Michel, and Andy van Dam for their tireless contributions to developing the system. We would also like to thank Peter Heywood, Scott Buchanon, and Chris Scott for the many hours they spent constructing the biology materials, and George Landow, David Cody, Glenn Everett, Rob Sullivan, Katherine Stockton, and Suzanne Keen Morley for producing the enormous corpus, *Context 32: A Web of English Literature*.

The work described in this paper was sponsored in part by a joint study contract with IBM, a grant from the Annenberg/CPB Project, and a research agreement with Apple Computer.

References

1. J. Conklin, "Hypertext: An Introduction and Survey," *Computer*, Sept. 1987, pp. 17-41.
2. N. Yankelovich, M. Meyrowitz, and A. van Dam, "Reading and Writing the Electronic Book," *Computer*, Oct. 1985, pp. 15-30.
3. D.C. Smith et al., "Designing the Star User Interface," *Byte*, Byte Publications Inc., Apr. 1982, pp. 243-281.
4. A. Goldberg, *Smalltalk-80: The Interactive Programming Environment*, Addison-Wesley Publishing Co. Inc., Reading, Mass., 1984.
5. B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Co., Inc., Reading, Mass., 1987.
6. N. Meyrowitz, "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework," *Proc. OOPSLA 86*, Sept., 1986, pp. 186-201.
7. B. Cox, "Message/Object Programming: An Evolutionary Change in Programming Technology," *IEEE Software*, Jan. 1984, pp. 50-61.
8. B. Stroustrup, *The C++ Programming Language*, Addison-Wesley Publishing Co., Inc., Reading, Mass., 1986.
9. K. Schmucker, *Object-Oriented Programming for the Macintosh,* Hayden Book Co., Hasbrouck Heights, N.J., 1986.
10. L. Tesler, "An Introduction to MacApp 0.1," Apple Computer, Inc., Cupertino, Calif., Feb. 14, 1985.
11. W.O. Beeman et al., *Intermedia: A Case Study of Innovation in Higher Education*, assessment report prepared for the Annenberg/Corp. for Public Broadcasting Project, Oct. 1987.



Bernard J. Haan, a research software engineer at the Institute for Research in Information and Scholarship at Brown University, is one of the original members of the Intermedia project team. As project leader of the programming team developing Intermedia's linking capabilities, he has been involved in all phases of the program's development, from design to implementation.

Haan is a graduate of Brown University, with degrees in computer science and French literature. He is a member of the ACM.

Norman K. Meyrowitz, associate director of the Institute for Research in Information and Scholarship at Brown University, has directed the Institute's hypertext and multimedia research program since he helped found IRIS in 1983. Most recently, Meyrowitz has managed and has been the principal architect of the Intermedia project. His major research interests are in the areas of multimedia documents, text processing, user-interface design, and object-oriented programming.

Meyrowitz graduated from Brown University in 1981 with a degree in computer science. He is a member of the Computer Society of the IEEE, the ACM, and Sigma Xi.

VICE PRESIDENT FOR INFORMATION SYSTEMS

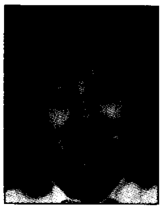
Adelphi University seeks a Vice President for Information Systems, reporting directly to the President. Duties include: information resources, systems and technology; establishment of policy and guidance of operations for educational technology and computing. High priority will be given to academic needs, with emphasis on enhancement of faculty and student utilization.

The successful candidate should have a proven record of leadership, and significant recent University teaching experience at the rank of full professor or its equivalent.

Salary competitive. Reply before February 15, 1988 to Chairperson, VPIS Search, Office of the President, Levermore 101, Adelphi University, Box 701, Garden City, New York 11530.

Adelphi UNIVERSITY

Adelphi University is an EO/AA Employer M/F



Nicole Yankelovich, a founding member of Brown University's Institute for Research in Information and Scholarship (IRIS), is project coordinator at the Institute. She is responsible for overseeing a number of projects, including experiments to introduce Intermedia into Brown University courses. She has participated in the design and development of Intermedia from its inception, contributing most heavily to the user-interface specifications.

Yankelovich holds the BA degree from Brown University in political science and is a member of the Computer Society of the IEEE, the ACM, and a number of educational computing organizations.

Readers may write to the authors at IRIS, PO Box 1946, Brown University, Providence, RI 02912.

Steven M. Drucker is a PhD candidate in the Brain and Cognitive Sciences Dept. at the Massachussetts Institute of Technology, studying robotics, tactile sensing, and machine learning as a Poitras fellow at MIT's Whitaker College. Previously, he spent two years at the Institute for Research in Information and Scholarship at Brown University, working first on an Intermedia prototype and later on the framework for the current version of Intermedia.

Drucker received the ScB in neurosciences, summa cum laude, from Brown University.