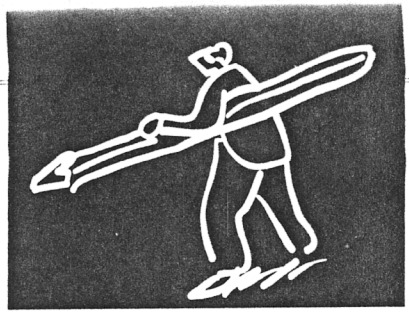


Completing the Job of Interface Design

JIM RUDOLF, Ergon Informatik
CATHY WAITE, Turing Institute

HyperNews lets you separate application and interface design to link a new interface to an old application with very little programming.

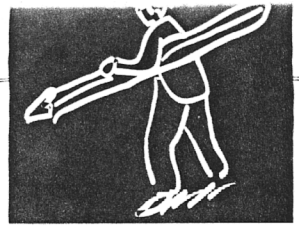
With today's powerful workstations and high-resolution bit-mapped displays, developers can take advantage of windows, mice, and menus to create a more friendly user interface. Unfortunately, while the user's workload goes down, the developer's goes up. Graphical user interfaces can be complex and require a surprising percentage of the application's total development effort.[1] Moreover, people without programming skills, like graphic artists and cognitive psychologists, are becoming involved in interface design, which requires that the design be more accessible to nonprogrammers.

User-interface management systems are generally used to support the development and execution of applications with interfaces. UIMSs vary in the number and applications with range of functions described. Some give only runtime support; others support the specification, design, implementation, execution, and evaluation of the interface.

In the old days, only programmers could develop GUIs because development was done exclusively with a textual specification language, which was then compiled and linked to the application. To make matters worse, there was no runtime support for communication or graphics. Fortunately, things have changed a great deal. Modern UIMSs support rapid prototyping using direct manipulation, so little textual programming must be done. With the advent of object orientation, newer generation UIMSs now define widgets as objects that can draw themselves and communicate with each other. Thus, an entire complexity layer is hidden from the interface designer.[2]

HyperNews, whose development is

0740-7459/92/1100/0011/$03.00 © IEEE



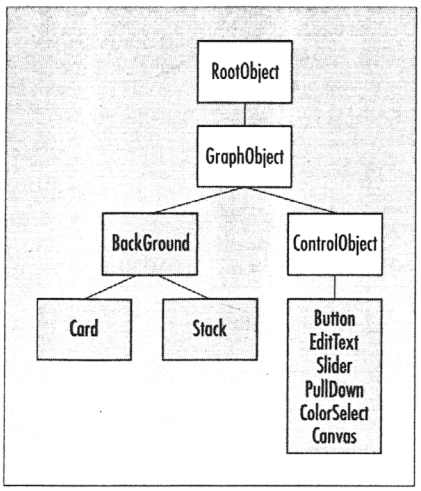


Figure 1. *The class hierarchy of HyperNews predefined object types. Classes in white boxes are abstract superclasses. Shaded boxes contain objects that can exist in a user interface.*

briefly outlined in the box on the facing page, is one of these newer generation UIMSs. HyperNews draws its design model from HyperCard, following HyperCard's application structure of stacks and cards. This structure simplifies GUI design and produces an interface that is easier for users to learn. HyperNews differs from HyperCard, however, in a number of ways:

♦ HyperNews was developed in News, a network windowing system that is a superset of PostScript. As a result, HyperNews has PostScript's powerful, device-independent graphics model. Post Script's graphics capabilities, important for rendering on high-resolution bit-mapped displays, are much greater than those of HyperTalk, HyperCards's scripting language. PostScript also supports the importing and exporting of device-independent graphics in ASCII.

♦ While both HyperCard and HyperNews support C, HyperNews also supports Prolog and Lisp, languages used at the Turing Institute.

♦ In some instances, HyperCard must access the Macintosh operating system, which is not a trivial operation.

HyperNews supports the rapid prototyping of GUIs by using specialization levels to group changes according to complexity. Many levels use direct manipulation to reduce the emphasis on programming, making quick, iterative design possible.[3] Thus, a variety of users can test many alternatives in a short time.

Those who want the flexibility and power to create more elaborate interfaces can do so by writing scripts. HyperNews incorporates a flexible interface-to-application linking model that is suitable for a range of application requirements.

HyperNews can be used to implement both the interface and the application, or to define a new interface for an existing Unix application. Thus, the same environment can generate interfaces for new applications as well as for existing applications with a command-line interface.

The emphasis in HyperNews is on flexibility. Interface developers with diverse skills can create GUIs using different levels of customization, resulting in different degrees of sophistication. GUI consistency is encouraged, but not enforced. Likewise, a number of application linkage models allow the use of a HyperNews interface on a variety of programs requiring different levels of support from the interface.

CREATING THE INTERFACE

Much of a HyperNews interface can be designed without writing any code at all. With direct manipulation, you can design a GUI simply by creating, moving, and resizing objects on the screen. Experimentation with different interface styles is possible with less effort in HyperNews compared to specification languages and tool kits, and immediate feedback can result in extremely fast development.

> Much of a HyperNews interface can be designed without writing any code at all.

Design model. The stack-based design model consists of individual windows that represent a stack of items holding different pieces of information. It is not unlike a pile of index cards, in which the top card is the only one visible.

To display different information, the cards are rotated to move a new card to the top. Cards can share a background, which holds elements common to each. A background might consist of graphics to be displayed for each card or labels shared by text fields.

Objects used for interaction, like menus and buttons, are called control objects. Control objects can reside on a card, background, or stack (objects on a stack are shared by all cards). If a label or graphic will be shown for a subset of cards, it should be placed on a background that the cards in the subset share. Control objects shown on every card, like a help button, should reside on the stack.

HyperNews objects. Designers use a number of objects when building a GUI, some of which are used often but are not trivial to implement, even for seasoned programmers. These objects are reusable in HyperNews, which keeps designers from having to reinvent the wheel and design common objects from scratch. The objects provide basic functions. If these are not sufficient, you can create new objects by modifying existing ones.

Figure 1 shows the predefined object types in HyperNews. Each system object represents a class, and the class hierarchy defines class interrelationships.

The classes shown in white are abstract superclasses that simply define behavior and structure common to their subclasses.

♦ *RootObject.* Defines instance variables and nongraphic methods common to all HyperNews object classes.

♦ *GraphObject.* The sole subclass of RootObject, it defines graphical behavior shared by all objects.

♦ *ControlObject.* Defines the behavior shared by all the interface objects that can exist on a stack. Its subclasses provide a set of predefined objects that will satisfy the basic needs of interface developers. Figure 2 shows sample control objects in the NewObjects stack.

The nine shaded classes in Figure 1 rep-

resent objects that can exist in a user interface. These classes are likely to make up the graphical portion of an interface; each class has visual characteristics, and each can respond to user input. They are

- *BackGround*. Defines behavior for the background.
- *Card*. Defines behavior for an individual card.
- *Stack*. Defines behavior for a stack.
- *Button*. Objects in this class include a pushbutton that inverts when you click on it; a checkbox, which you can toggle on or off; a drawing button, which can have one or more HyperNews drawings pasted into it; and a transparent button used to create a transparent field that detects input.
- *EditText*. Represents text fields. You can use objects in this class, which have varying attributes, to create static text fields and editable fields with or without a scroll bar.
- *Slider*. Provides scroll bars and percentage-done indicators.
- *PullDown*. Provides menus.
- *ColorSelect*. Defines the color for an object's attribute, such as the stroke and fill colors for a button.
- *Canvas*. This class is a special case. Its main purpose is to provide access to a region of the screen and to let existing applications incorporate a HyperNews interface with few changes other than to the application's I/O routines. Unlike the other subclasses of ControlObject, a Canvas object has very little predefined behavior. The HyperNews terminal emulator is an example of an application that uses a Canvas object.

HyperNews's inheritance between object types is different from HyperCard's approach. In HyperCard, there is no inheritance, and similar methods are not shared among types. HyperNews also includes more control object types and abstract superclasses.

Implementing the interface. To design a HyperNews interface, you begin with a stack. Essentially, a stack has the same role as a window — to define a movable, possibly resizable area of screen that represents an application. A HyperNews application will consist of one or more stacks, just as other window-based applications will have one or more windows.

To create a new stack, you simply change an existing stack and save it with a new name. A template called the Untitled stack is provided as a suggested starting point. This template provides a shape for the stack, a text field for the stack name, an iconify button, and navigation buttons to move among cards. If you'd rather not use it, you can create a different stack and copy it just as easily.

Once you create a stack, you choose a control object that has the desired appearance and behavior. (If one doesn't exist, you'll have to choose the next best thing and customize it later.) It may reside on the NewObjects stack or on some other, already created stack. You then copy that object from the original stack and paste it to the new stack, where you can then change its appearance or behavior if desired.

HYPERNEWS TO HYPERLOOK

HyperNews development began in 1987 at the Turing Institute, and the first internal release was completed in 1988. A later release was made available for nonprofit use in 1990. It has been used for various commercial and research ventures, including at least four ESPRIT projects.

HyperNews continued to evolve until spring of this year, when the Turing Institute made it available commercially. The new product, called HyperLook, is a more robust and commercial-quality version of HyperNews. A single-user license is $1,095. Contact the HyperLook group, 44 (41) 552-6400 fax: 44 (41) 552-2985.

HyperNews was written entirely in the News windowing system. In the current version, the kernel consists of roughly 11,000 lines of code, with an additional 10,000 lines of code in the form of stacks. It requires OpenWindows, and runs on Sun Microsystems workstations.

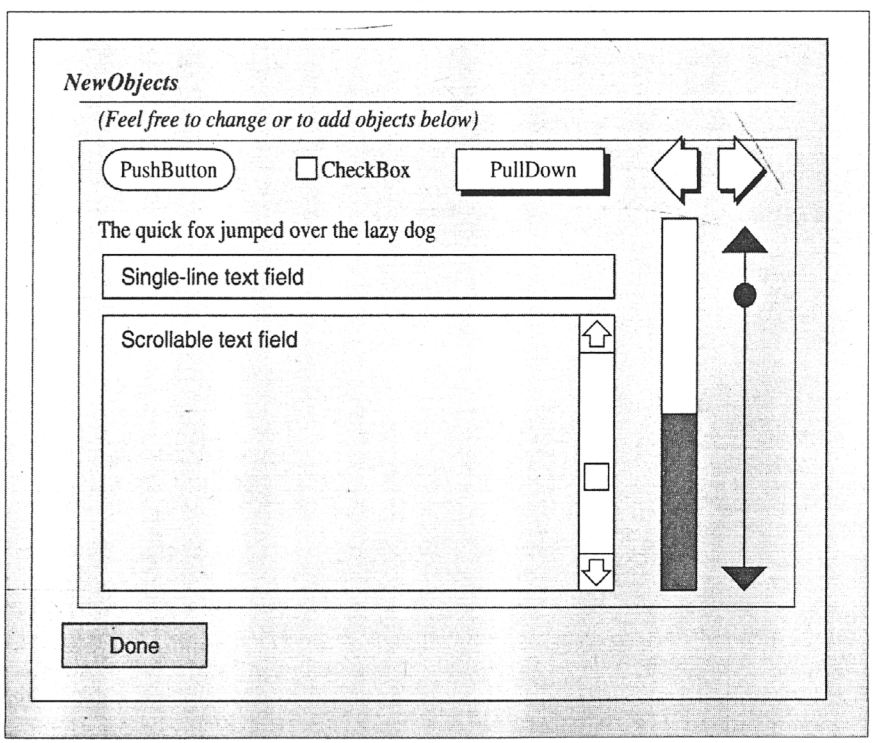
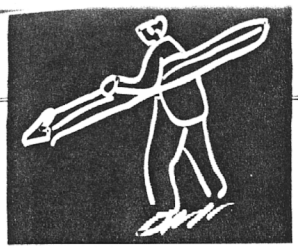


Figure 2. The NewObjects stack.



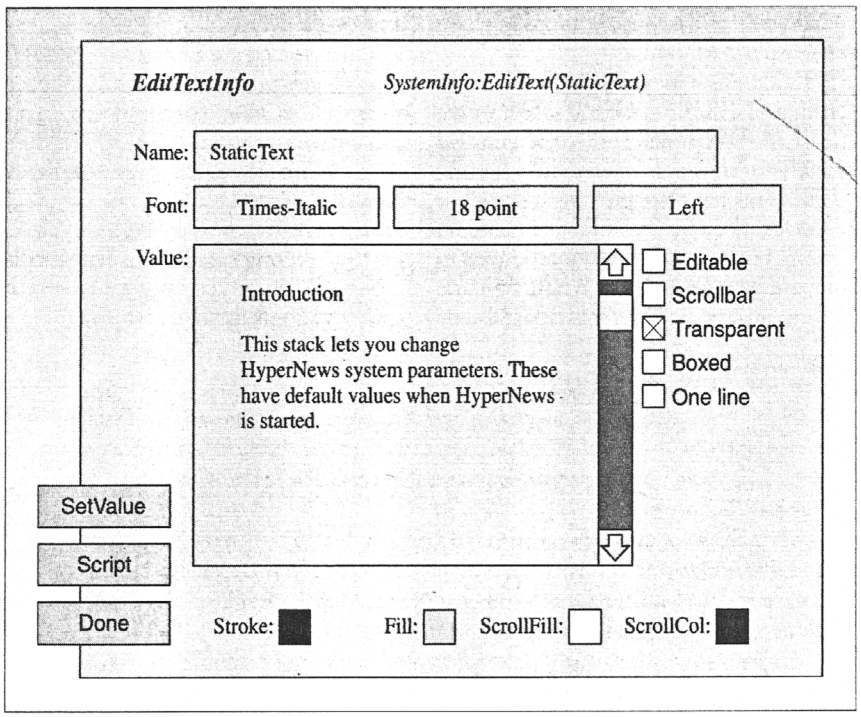


Figure 3. The info stack for EditText objects.

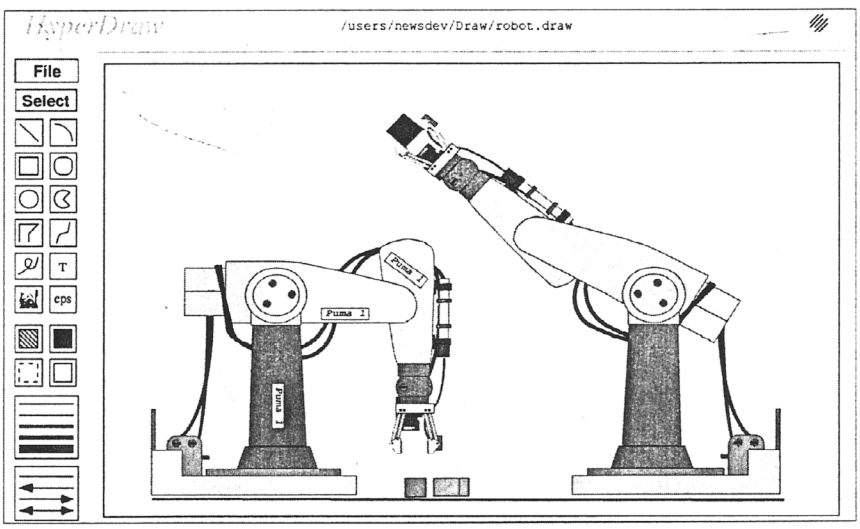


Figure 4. The HyperNews drawing tool.

A control object pasted onto a stack will always be placed on the current card. From there, you can move the control object to the background of the current card or to the stack, where it will always be visible.

The process of copying and pasting objects to create new ones is often referred to as prototyping.[4] The existing object is considered a prototype on which a new object will be modeled. Prototyping differs from instantiation, in which a message calle[illegible]w" is sent to a metaclass to create a new object. For many users new to object-oriented philosophy, the concept of prototyping is easier to comprehend than instantiation. To create a new scroll bar, just select one, copy it, and paste it.

Thus, HyperNews lets you duplicate objects across stacks. You can transfer previously developed interfaces and their objects from one stack to another in a straightforward manner. Using objects from an existing stack, you can create a similar look for the new stack, you providing consistency simply through copy-and-paste operations.

Specialization without programming. A copied and pasted object is considered to be in the same class as the prototype object until you modify it. Once you do, the new object becomes specialized and effectively defines a new object class, which is a subclass of the original object. In this way, you can make an object specialized and then prototype it many times to create multiple copies of an object class you define.

Although you need some programming knowledge for more elaborate stacks, you can do a surprising amount of specialization without programming. Different ways of modifying an object are broken into specialization levels. At the higher levels you can use direct manipulation without programming; at the lower ones, you must use the scripting language (PostScript).

Editing the stack. The highest specialization level is simple stack editing, which involves deleting or adding objects or changing their appearance. When a stack is in edit mode, you can create or delete new cards and backgrounds, add new control objects, or delete or modify existing control objects by moving or resizing them with the mouse.

Changing object attributes. The next level of specialization involves changing the values of object attributes. accessible via an object's "info stack." Every object has a number of attributes that affect its appearance. Typical attributes include the object's color and the font style and size used for displaying its text.

Figure 3 shows the info stack for EditText objects, called the EditTextInfo stack. It has a number of flags that determine if the field is outlined with a box, if it has a scroll bar, and if a user can edit the text. It also has four ColorSelect objects along the bottom, which define the colors used to display different areas of the EditText object. Other info stacks display similar object attributes that you can modify with a mouse click or a few keystrokes.

The info stack concept is similar to that used by HyperCard, except that you can specify a link in a HyperCard button's info dialogue. By pressing a button, the user can then create a link to another card. In HyperNews, you must write a one-line script to make a link to another card.

Elaborate visual changes. You can use an object's info stack to modify certain visual attributes, but the changes you can make from the info stack are limited. To make more elaborate changes to the appearance of stacks and buttons (at the next level of specialization), you need the HyperNews drawing tool, called HyperDraw, which is shown in Figure 4.

HyperDraw, a direct-manipulation graphical editor, is used to create all the graphics in HyperNews. It provides facilities for creating text and graphics using the mouse and keyboard. Besides being a complete drawing tool, HyperDraw connects the interface to the graphical world outside HyperNews. It can export graphics into a PostScript file and import digitized images and graphical descriptions in a number of formats. Once graphics are imported into HyperDraw, you can use them for the same purposes as any drawing created within it.

In HyperCard, graphics are created with a paint tool, implemented as a tear-off menu that can be visible at all times. While some of its operations are more sophisticated than those offered by the drawing tool, it cannot easily import or export other graphics formats.

Creating stack graphics. At the next level of specialization is the creation of stack graphics. To create graphics for a stack object, you develop them in HyperDraw and then paste them into the stack. The new graphics replace the existing stack graphics, and the new stack shape is defined by the outline of the new graphics.

To alter the appearance of a stack, you may only want to change the color scheme, or you may want to do much more, such as adding complex graphics and changing the shape of the stack. To add graphics, you create them in HyperDraw, copy them to the Clipboard stack, and then paste them into the stack being modified.

HyperNews lets you transfer previously developed interfaces and their objects from one stack to another.

You are not restricted to a rectangular stack shape. Unlike most other windowing systems, News supports canvases, and thus HyperNews stacks, of arbitrary shape. The EditTextInfo stack in Figure 3, for example, could not have been implemented in most windowing systems because of its protruding buttons in the lower left corner.

Figure 5 shows another example of a nonrectangular stack. To define this stack, you first create four rectangles with a horizontal line near the top within HyperDraw. You then copy the drawing to the Clipboard stack and paste it into an existing stack. Finally, you add text using EditText objects, and add graphics by pasting drawings into Button objects.

The outline of the stack is defined as the outer boundary of the graphics that make up the stack. Using this technique, you can create a variety of unique stack shapes.

Button graphics. Another way HyperDraw can be used to modify the appearance of a GUI is to create button graphics. As you do for stack graphics, you paste a drawing created in HyperDraw into a button via the Clipboard stack. The drawing can con-

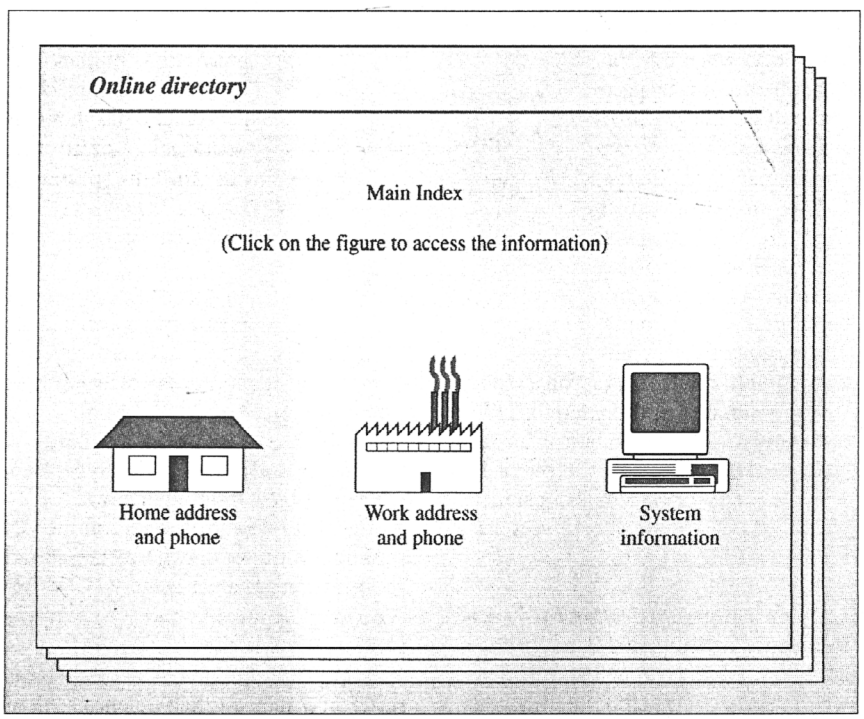


Figure 5. A specialized stack shape.

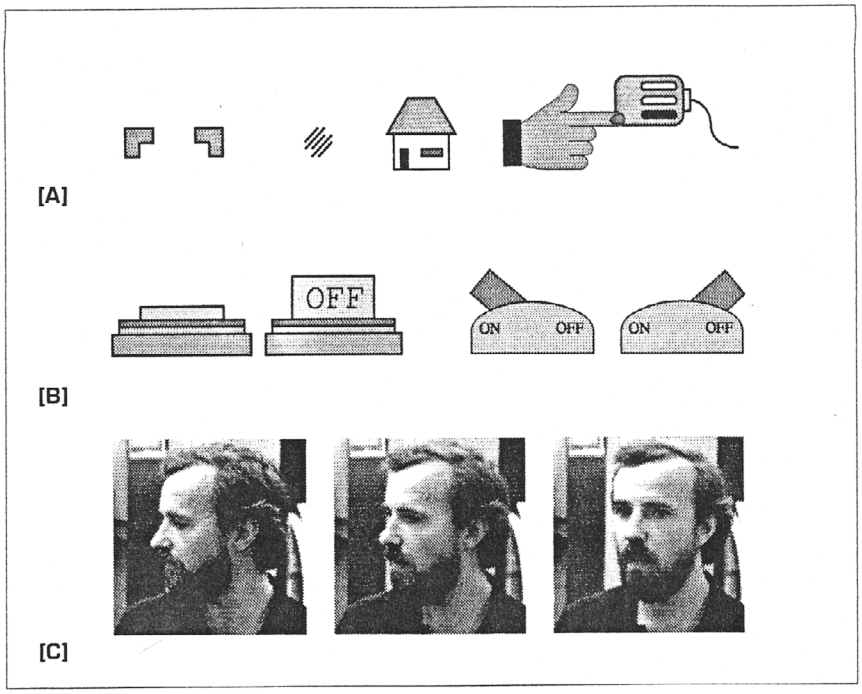


Figure 6. Specialized buttons. (A) Drawing, (B) sequence of drawings, and (C) sequence of images.

sist of encapsulated PostScript or digitized images that have been imported, as well as graphics created within HyperDraw.

Figure 6 shows examples of drawing buttons. Figure 6a shows different buttons with single drawings, including resize buttons used in the corners of resizable stacks and an iconify button.

Figure 6b shows examples of two groups of drawings that can be pasted into the same button for a cycling effect when the button is clicked on. By pasting multiple images into a button, you can generate an animated sequence. Figure 6c shows three digitized images that can be pasted into a drawing button in a sequence similar to that for drawings.

In contrast, HyperCard graphics are created on cards and backgrounds using the paint tool. The primary way to customize a button is to associate an icon with it. If the set of icons provided with the system is not sufficient, you must use Macintosh resources outside HyperCard.

Specialization with programming. To make more substantial changes to an object's appearance or to add behavior, you must write a script that will define new methods for the object. An object's script is a set of methods written in PostScript. The methods are local to the object and are invoked when the object receives a message of the same name.

PostScript is a stack-based language with a postfix syntax. In HyperCard, scripting is done with HyperTalk, an easy-to-learn language with a flexible syntax. However, although PostScript is more difficult to learn, it has built-in graphics operators that can create sophisticated graphics to let you customize object appearance. It also has conventional operators for customizing behavior. HyperTalk lets you customize only behavior.

HyperNews lets you customize both object behavior and appearance.

Appearance. By redefining the routine that determines how an object is drawn, you can dramatically alter an object's appearance. To do this, you redefine the Draw method, defined in every class and invoked by HyperNews when the object must be redrawn.

For example, the drawing routine for a pushbutton object draws a rectangle with rounded corners and a centered label. You can override this method by defining a Draw method in the object's script. If you wanted to, you could write a method that reads an instance variable denoting temperature and use that method to determine what shade of red or blue to redraw the button.

Behavior. The behavior of most control objects is initiated when you click on the mouse. Behavior is determined according to the ways the object responds to user input. When you click on a control object, HyperNews sends the Action message to the object. If you do not define an Action method for the object, the message is ignored.

You can change an object's behavior or supplement it by adding methods to its script. Generally, object responses fall in two categories, semantic and feedback. By design, HyperNews provides only feedback. An example is a button inverting or a menu expanding when you click on it. HyperNews acknowledges user action but does not act on it. Semantic actions are left for you to define in the Action method, because each object will most likely have application-specific behavior beyond the scope of HyperNews.

Adding system attributes. You can create new methods and instance variables in PostScript to specialize an object type. The new structure and behavior are readily accessible from the script, but only if you have some programming experience.

Since a new attribute defined in an object will probably be represented by an instance variable, you should be able to modify the instance variable using a mouse click. But this will require a more powerful type of specialization when you define a new object class whose object attributes can be set using direct manipulation through the info stack.

To illustrate, suppose a programmer defines a new Boolean instance variable called HalfSize for a subclass of Button that would render the object at half its normal size. The Draw method is redefined to check this attribute and draw the object at half size if the instance variable is true.

This feature would be easy to use, except that a graphic designer would have trouble accessing the attribute's setting. To remedy the situation, the programmer could copy the ButtonInfo stack, renamed perhaps HalfSizeButtonInfo, and then

specialize it. An additional checkbox could represent the current state of the HalfSize instance variable, which a designer could read and modify using the mouse. Future users will not know if the system defined this object class or if it is a specialized subclass. Thus, any system extensions will be seamless. As the system evolves, changes will be completely integrated.

By combining the two specialization techniques of direct manipulation and scripting, HyperNews provides a flexible and extensible interface development system. Not only can you design much of the interface without programming, but you can define new object types with new attributes that can be modified through direct manipulation alone.

COMMUNICATION CAPABILITIES

Once you have designed the interface, you can run the resulting stacks in HyperNews. At runtime, HyperNews provides communication capabilities among all HyperNews objects, such as control objects, cards, backgrounds, and stacks. Also, each stack can communicate with a client — a Unix application in C, Prolog, or Lisp — through HyperNews's high-level-language interface.

Communicating among objects. Objects in HyperNews communicate by passing messages. When an object receives a message, it tries to execute a method with the same name as the message. Standard methods are defined for each object, and you can define new methods using scripts. HyperNews translates input events, such as keyboard presses or mouse clicks, into messages and passes them on to the appropriate object.

The route that messages take when they go unanswered is defined by the HyperNews message hierarchy, shown in Figure 7. If an object doesn't understand a message, it passes the message to its parent, as defined by the message hierarchy. The hierarchy follows a front-to-back order, with control objects to the front, and the stack (or a client, if one exists) the farthest back.

For example, if a button gets a message it doesn't have a method for, it will pass the message on to the card. Likewise, the message will be passed on to the background, the stack, and the client (when one exists), until a receiver for the message is found. If none of the objects in the hierarchy can process the message, it is discarded. This technique is called unreliable message passing, because there is no guarantee that a message can be handled by an object. Because a message that goes unhandled will not raise an error condition, you can include message sends to objects that do not exist (or a client yet to be implemented), without constructing test stubs. When you add the new objects (or attach the client), the messages will automatically be handled.

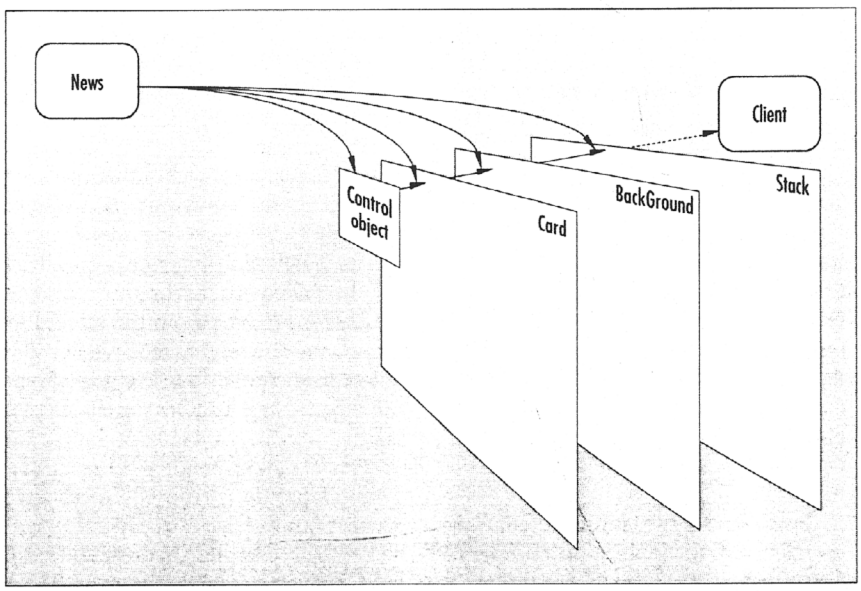


Figure 7. The HyperNews message hierarchy.

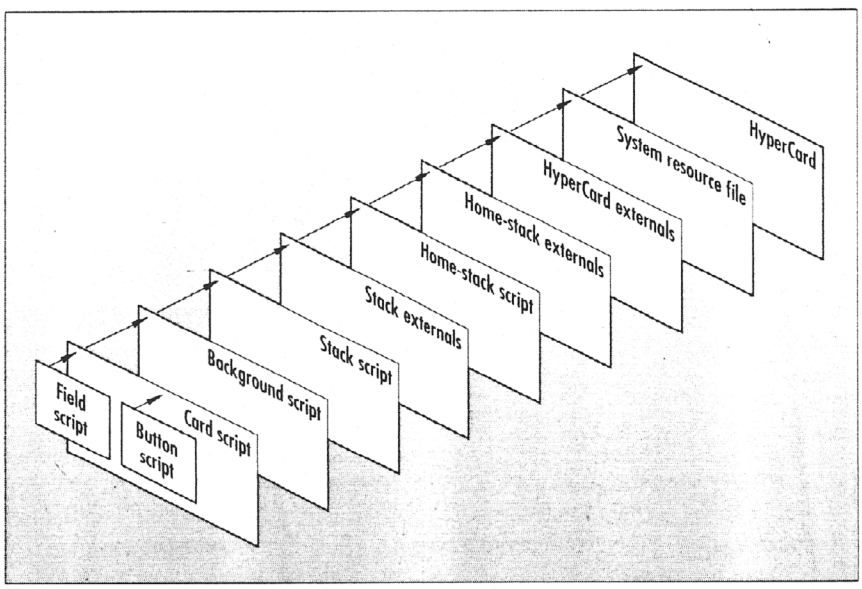
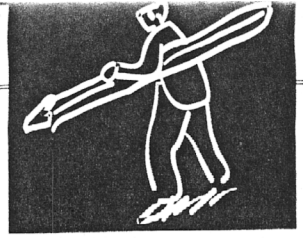


Figure 8. The HyperCard message hierarchy.



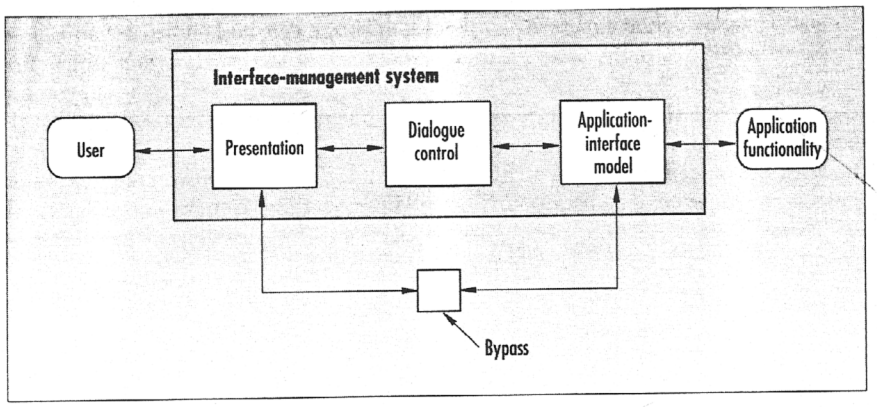


Figure 9. The Seeheim Model of a user-interface management system.

Figure 8 shows the HyperCard message hierarchy, which is similar to HyperNews's structure; a message travels front to back until an object can handle it. However, HyperCard does not support unreliable message passing. All messages must be handled, or an error is reported. Additionally, the HyperCard application is at the top of the message hierarchy, with clients (in the form of external functions), between the stack and HyperCard.

Communicating with clients. Although you can develop an application completely within HyperNews, interfaces in Prolog, C, and Lisp are also provided for applications not suited to PostScript, for programmers not willing to learn PostScript, or for communicating with clients. These interfaces provide an effective way for the different parts of the application to communicate with the HyperNews interface.

Each HyperNews stack can have one client — a Unix process — associated with it, and a client can be connected to multiple stacks. When the client is connected to a stack, it can communicate with any object in the HyperNews address space by specifying the names of the receiving object and the stack on which it resides. Objects can communicate with another stack's client by sending the stack a message that it cannot process. Clients are typically activated when the associated stack is loaded. However, a client can load a stack and create a link to it at any time, as long as HyperNews is active.

After connecting to the HyperNews environment and a stack, a client will register interest in receiving certain messages from the stack. Registering interest means that an application function, or callback, will be called whenever the registered message is received from the stack. If an unregistered message is received, it will be ignored. This policy is in keeping with the way HyperNews handles unreliable messages.

While not responding to a message from HyperNews, the client waits for events from the stacks to which it is connected, and executes the associated callback when a message is received. When more than one callback routine is registered with each message, the routines are executed in the order in which they were registered.

Sending a message from a client to an object is similar, except that the message need not be registered. The client specifies the HyperNews object by specifying the stack name and the object on the stack, the message to be sent, and the arguments to be sent with it.

LINKING TO THE APPLICATION

Figure 9 shows the Seeheim model[5] of a UIMS, which defines the logical structure of the UIMS's runtime component and hence models how the UIMS communicates with the application functions. Presentation deals with the external presentation of the interface, and generates the images that appear on the screen. It also deals with the input devices, converting raw user input into forms other interface components can use. Dialogue control defines the structure of the dialogue between the user and the application. Depending on input from the presentation or application, dialogue control defines the state of this interaction or dialogue.

The third and generally least developed component of the Seeheim model is the application-interface model. This component mediates between application functions and dialogue control. It lets you explicitly specify the communication protocol between the UIMS and the application.

The fourth component, sometimes called the bypass, lets the application-interface model and presentation communicate directly. The bypass is used only under exceptional circumstances, and many believe that dialogue control should not be short-circuited in this manner. Doing so can create fuzzier borders between components and make it harder for the design to be modular. For this reason, we do not describe the bypass further.

With these components, you can link HyperNews interfaces to internal applications, which were developed within HyperNews, to external applications, or to existing applications developed without HyperNews. You can also link to more than one application.

Linking is done primarily through the application-interface model. The model can be only a conceptual link of application and interface, or it can be an independent runtime component that mediates between the two. How much you can separate the interface from the application depends on how concrete the application-interface model is.[6] The more concrete the model, the easier it is to develop the interface and application independently.

Not all applications benefit from being separate from the interface. In some application domains, such as drawing tools, the application *is* the interface, and system performance may decrease if you were to separate the two. For the most part, however, there are many advantages to maintaining a strict separation. One is that designers without programming skills can design interfaces, while programmers design and implement the application. If the

application-interface model is concrete with a well-defined communication protocol, designers can experiment with and develop several interfaces for the same application, freely interchanging them.

Internal applications. When the application and interface are linked only conceptually, you can implement the application entirely in HyperNews. You add functions to HyperNews objects merely by writing a script. The application-interface model exists only as the set of methods that define the functional core. Figure 10 shows HyperDraw as an example of an internal application-interface model. Each object in HyperDraw has scripts added to define its behavior. Most of the functionality is in the Canvas object, which knows how to draw, manipulate, and save PostScript graphics.

A tightly coupled application and interface is good for applications like drawing tools because an effective drawing tool is highly interactive and provides instantaneous user feedback. The performance of the system would decrease dramatically if the interface had to communicate with an external client while the user is interactively resizing or moving a component.

One way you could improve the performance of such applications is to add more information to dialogue control so that it knows about the drawing objects and is able to resize or move them without communicating with the application. With an interactive drawing tool, however, you would have to move virtually all application information into the interface because the interactive aspects more or less constitute the entire application. The application would be left with only the functions of saving the drawings to, and reading from, external files.

Thus, to develop a drawing tool in HyperNews, you would want to write it directly in PostScript. Although by doing so, you can't physically separate the interface and the application, neither can you separate them conceptually, so physical separation would not be wise anyway.

You can also define an application internally in HyperCard by adding scripts (in HyperTalk) to objects. For an internal application-interface model, HyperNews and HyperCard act in much the same way.

External applications. The internal application-interface model requires programmers to learn PostScript and work in HyperNews. When this is not desirable or possible, interfaces can be built using the client interface without programming in HyperNews at all.

These external models sacrifice some efficiency because all interactions must be passed from the GUI to the application and back again. But the reward is a cleaner separation between the application and interface, which means that interface designers have a much easier job. By using direct manipulation and being aware of the messages sent when the user interacts with each object, they can construct an interface. There is no need to learn a new programming language or have in-depth knowledge of HyperNews. Moreover, the sacrifice in efficiency is not serious; the results are certainly adequate for prototyping a design.

Figure 11 shows the external application model. The structure reflects the development of the Optimist system,[7] which was funded by an oil company to assist geologists in oil exploration. The application-interface model, a Unix process, exists outside HyperNews and is bundled with the application functions. The model again exists conceptually — not as a set of methods, as in HyperDraw — but as the mapping from HyperNews messages to application functions within the application process.

Optimist is a cooperative expert system that acts as a colleague. The user and Optimist interact to assess the probability of finding oil at a prospective site. The system argues its case on the basis of consistency with previous decisions, and the user supplies extra knowledge.

These discussions are highly interactive, making a good user interface essen-

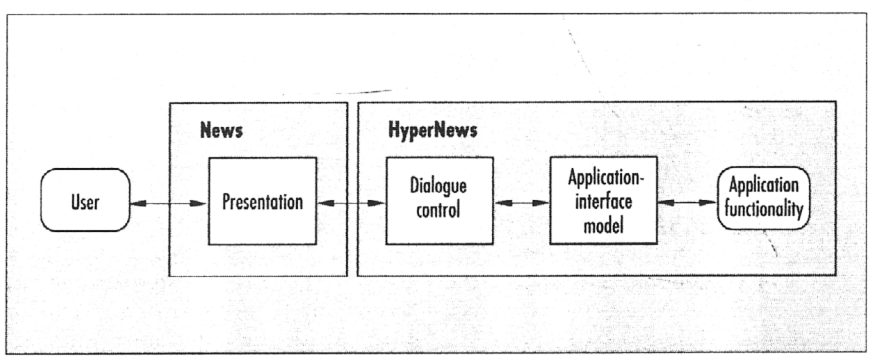


Figure 10. HyperDraw: the internal application model.

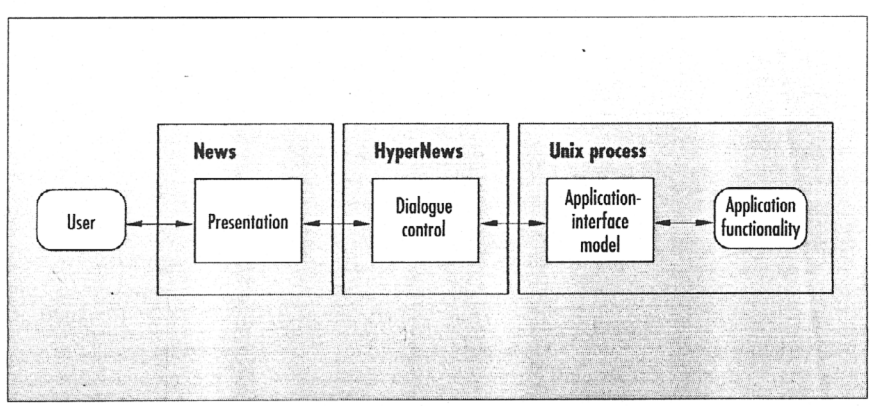


Figure 11. Optimist: the external application model.

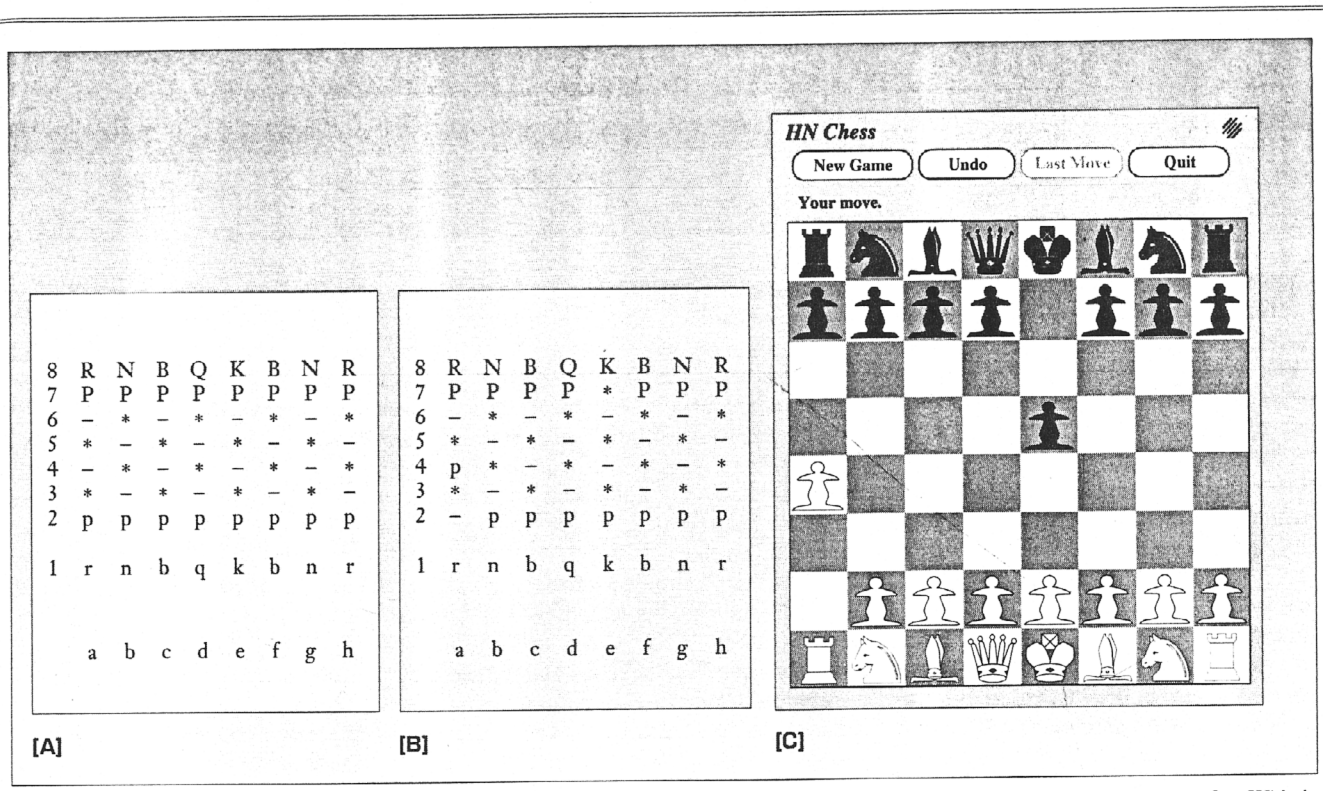


Figure 12. *(A) The Berkeley Unix chess interface, (B) the same interface after the user and the system have moved once, and (C) the HyperNews interface. With the command-line interface, the user must type letters and numbers to move; the HyperNews interface is much more intuitive and enjoyable.*

tial. The developers of Optimist's interface used only direct manipulation editing and attribute specification. They did not add any behavior to objects (did no scripting). They exploited HyperNews's message-handling approach to communicate with the client, a Prolog program that handled all messages either by executing some application code or by sending messages to other HyperNews objects. Because all sent messages move up the message hierarchy and, if not handled, end up being passed to the client, the developers could simply name objects to program the client. Even the Action message, sent when the user clicks on a button, works its way up the hierarchy to the client.

This ability to link to an external model is a feature unique to HyperNews. In Hypercard, for example, external application functions are implemented as external functions or commands — single routines in C, Pascal, or assembly language. (In HyperNews, you can't write single routines and access them as stack methods) The designer registers routines as resources with either the stack or the HyperCard application itself, using a tool such as ResEdit. Conceptually, these routines are additional methods and must be executed in the HyperCard environment. Even though these external routines look like and are invoked in the same way as HyperCard's own functions, they are not considered stand-alone applications. A group of routines may form an application's functional core, but the routines are independent with no shared state or process.

In HyperNews, on the other hand, all methods have access to the News address space, in which a shared state can reside. We are not saying that you can't access application functions from HyperCard; you just can't connect a separate application to a HyperCard stack. As an alternative, you can invoke external functions written in a high-level language and implement all the application functions using these external commands. The messages sent to buttons, fields, etc. will be passed up the message hierarchy and handled by the external functions. Thus, as in HyperNews, you don't have to write scripts when using external functions.

Existing applications. In the previous application-interface models, the application had to have extensive knowledge of the interface, either because it was implemented in HyperNews (like HyperDraw) or because it implemented the behavior of each HyperNews object (as in Optimist). In both cases, this knowledge was necessary because the application-interface model existed only conceptually, rather than as a distinct concrete module.

The same low degree of application-to-interface coupling also makes it possible to link a HyperNews interface to an existing application with a command-line interface — which represents a concrete application-interface model — without changing the application. Although this link is not as efficient as one in which the interface and application are designed to converse with each other, it is certainly feasible to retrofit existing application interfaces in this way.

As an example, consider the Berkeley Unix chess application, which has a fairly primitive, command-line interface. The application displays the chess board as shown in Figure 12a.

The user indicates moves in a strict format, such as a2a4, which denotes moving the piece from square a2 to square a4. If the incorrect format is used (such as a2–a4), the application responds with "eh?" If a string is entered that can be interpreted by the application as a move, but is an illegal move, the application responds with "Illegal move." If a correct move is entered, the application accepts the input, echoes the move to confirm it, and prints out its own move:

```
1.a2a4      [User's move]
1.... e7e5  [System's move]
```

The user can press the carriage return key at any time to view the current state of the chess board: Figure 12b shows the current state after the above exchange.

This interface is difficult to use, for two reasons: it has a rigid input syntax, and it poorly represents the state of the board. Users have problems not only mapping their intentions onto the required input format, but also evaluating the system's actions. In this case, it does not matter how good the application's chess strategies are, users are apt to judge the application more on the intuitiveness of the interface. With HyperNews, you can fit a graphical interface onto this application without changing the application at all. In fact, that is precisely what the developers did.

Figure 12c shows the HyperNews chess interface after two moves. The user makes a move by dragging the appropriate piece to a new location. The interface responds by moving the outline of the piece to the location indicated by the user. This is translated into the format expected by the chess application. If the application echoes the move, the interface moves the piece to the new location; otherwise, the user will see "Illegal move." When the application makes a move, the interface updates the board accordingly.

With the graphical interface, the application is easier to use. Making moves is more intuitive to the user, and the representation makes it easier to evaluate moves. It is also impossible to make syntactic errors, like using the wrong format for the moves, which removes the need for the information-less "eh?" response from the application. The user can still make semantic errors, such as moving a piece to an illegal square. However, the interface uses the application to verify the move, only moving the piece when the application has confirmed it, and displaying a meaningful message otherwise.

As Figure 13 shows, the application interface model in this case is a separate Unix process that communicates with HyperNews using the C client interface. The model communicates with the chess application, another separate Unix application, by way of a Unix pipe. As far as the chess application is concerned, it is reading and writing from its standard I/O channels. The application-interface model takes the character output from the chess application and converts it into messages, which are sent to the HyperNews stack. The stack then relays the messages to the appropriate HyperNews object, removing the need for the application-interface model to know about the objects in the interface. In turn, messages from HyperNews objects are sent to the stack, which converts them to the appropriate character sequence as required by the chess application.

In this example the interface has no knowledge of the application in that it does not understand how to play chess. The only knowledge it has of chess is its own graphical representation of a chess board and the locations of the pieces. In addition, the application has no knowledge of the interface representation being used. It is easy to see how a different interface could be linked to the same application by changing the stack that the application-interface model communicates with. Alternatively, the stack could be used as the interface to a different application by linking it to a different application-interface model/application pair.

In HyperCard, you cannot link to an existing application because HyperCard does not support the concept of an external application, only the use of individual external functions. Even if it did support external applications, it still could not link to an existing application because the Macintosh operating system does not support the notion of pipes (found in Unix), which let you redirect the application's I/O.

Linking to multiple applications. You can also link a single HyperNews interface to more than one application. Although the

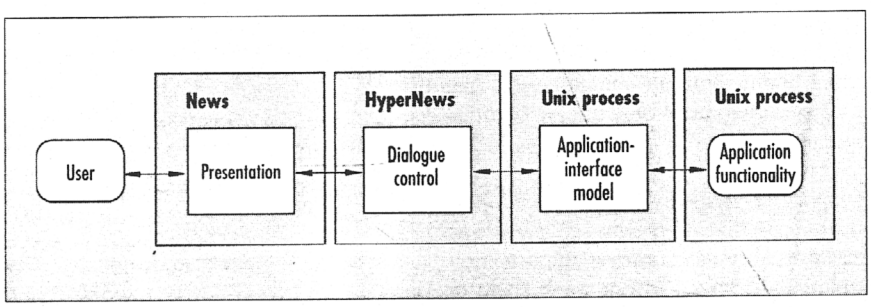


Figure 13. Chess: the existing application model.

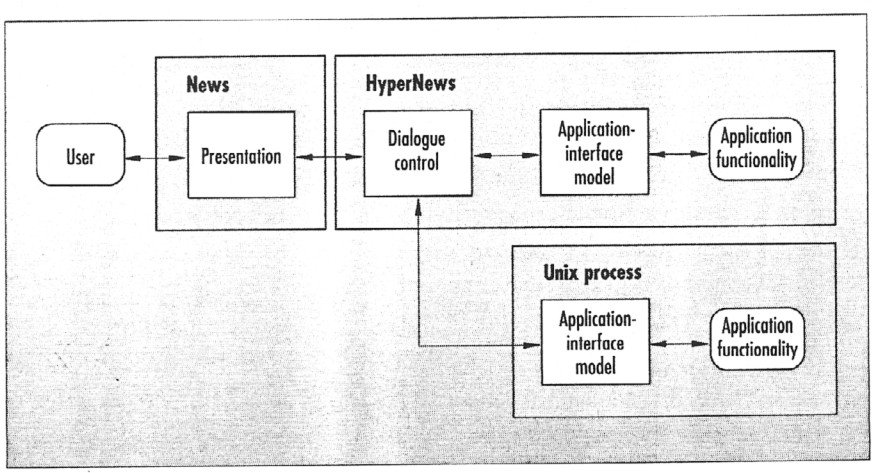


Figure 14. Integrated Fault Management Environment: the multiple-application model.

Seeheim model links to only one application core, there is no restriction to the number of applications that can be connected to a UIMS. For each application core, however, there must be an associated application-interface model.

HyperNews has been used in this way in the Integrated Fault-Management Environment,[8] an environment for building a qualitative model of an engineering system that must be diagnosed. There are two stages to the diagnosis: generating a model of the system, and executing the model to analyze it.

IFME, whose model is shown in Figure 14, is an example of a single interface that drives two separate application cores. The interface is connected to an application written within HyperNews that can be used to graphically construct the model. It is then compiled into an external application, which can be accessed with the same interface used to execute the model. From the user's point of view, interacting with two application cores is seen as two modes within the one application. The obvious advantage here is that the user has to learn only one set of interface concepts.

HyperCard is not flexible enough to support this model because it neither permits communication between stacks and an external application, nor supports multiple clients linked to one application or vice versa. HyperNews's flexibility, on the other hand, lets you develop interfaces for a range of applications, from simple to complex. This generality is what sets it apart from other UIMSs. Although HyperCard is popular as a prototyping environment for interface design, it is best suited to applications that are interface-intensive and have little functional core.

HyperNews has accomplished its primary development goal, which was to take some of the effort out of creating interfaces. GUI designers with backgrounds in graphic design or cognitive psychology instead of programming have a powerful UIMS that doesn't make them write programs. Instead they can use a copy-and-paste technique to intuitively construct an interface. Not only is it possible to generate and experiment with different interfaces to new applications, but within the same environment, GUIs can be generated for existing applications. Thus, applications need not be discarded if they have outdated user interfaces.

Through environments like HyperNews, contemporary interfaces can be added to existing applications to take advantage of the growing knowledge in interface design and user requirements. In the process, designers will produce a system that they find easier to use and more relevant to their task, whether or not the underlying application has changed.

As UIMSs continue to evolve, HyperNews should prove to be extensible enough to incorporate innovative new forms of interaction. Once HyperNews has been modified to provide a new type of interaction, designers should still be able to retrofit applications to exploit new interaction techniques without drastically changing the application. We feel this flexibility is due in large part to the flexibility on both sides of the interface: the side that involves GUI design and the side that links it to the application. ♦

ACKNOWLEDGMENTS

The principle architects of HyperNews are Arthur van Hoff and Tim Niblett. HyperNews was implemented by Arthur van Hoff.

REFERENCES

1. B. Myers, *Creating User Interfaces by Demonstration*, Academic Press, Boston, 1988.
2. D. Hix, "Generations of User-Interface Management Systems," *IEEE Software*, Sept. 1990, pp. 77-87.
3. E. Hutchins, J. Hollan, and D. Norman, "Direct Manipulation Interfaces," in *User Centered System Design*, D. Norman and S. Draper, eds., Lawrence Erlbaum Assoc., Hillsdale, N.J., 1986.
4. A. Borning, "Classes Versus Prototypes in Object-Oriented Languages," *Proc. IFIPS: Fall Joint Computer Conf.*, IEEE CS Press, Los Alamitos, Calif., 1986.
5. H. Hartson and D. Hix, "Human-Computer Interface Development: Concepts and Systems for Its Management," *ACM Computing Surveys*, Mar. 1989, pp. 25-92.
6. C. Wood and P. Gray, "User Interface — Application Communication on the Chimera UIMS," *Software: Practice and Experience*, Jan. 1992.
7. P. Clark, "Representing Knowledge as Arguments: Applying Expert System Technology to Judgemental Problem-Solving," in *Research and Development in Expert Systems, Vol. 7*, T. Addis and R. Muir, eds., Cambridge University Press, Cambridge, UK, 1990.
8. N. Ward, "AI and KBS for Space Applications," *Proc. European Space Technology Center Workshop*, Estec, Noordwljk, The Netherlands, 1991.

Jim Rudolf is a user-interface developer at Ergon Informatik. His research interests include graphical user interfaces, object-oriented programming, computer-supported cooperative work, and hypermedia. He was at the Turing Institute when the work described in this article was being done.

Rudolf received a BS from California Polytechnic University and an MS from Oregon State University, both in computer science. He is a member of the IEEE Computer Society, ACM, and SIGCHI.

Cathy Waite is a member of the research staff at the Turing Institute, where she is working on the development of X Window System interfaces for interactive vision-processing applications.

Waite holds an MSc in computer science from Strathclyde University. She is registered for a PhD in the application of design theory to user-interface design environments at the University of Glasgow.

Address questions about this article to the HyperLook group, The Turing Institute, George House, 36 North Hanover St., Glasgow G1 2AD, UK; Internet hyperlook@turing.com.