

# The Information Grid:

## A Framework for Information Retrieval and Retrieval-Centered Applications

Ramana Rao, Stuart K. Card, Herbert D. Jellinek, Jock D. Mackinlay, and George G. Robertson  
Xerox Palo Alto Research Center

### ABSTRACT

The Information Grid (InfoGrid) is a framework for building information access applications that provides a user interface design and an interaction model. It focuses on retrieval of application objects as its top level mechanism for accessing user information, documents, or services. We have embodied the InfoGrid design in an object-oriented application framework that supports rapid construction of applications. This application framework has been used to build a number of applications, some that are classically characterized as information retrieval applications, others that are more typically viewed as personal work tools.

### INTRODUCTION

Computer workspaces have long been dominated by a location-centric organization of application objects (e.g. files in directories or messages in folders) and a navigational metaphor that requires users to specify a path to access desired objects. An alternative paradigm growing increasingly common is the use of computer-supported retrieval to assist the user in finding objects associatively by their properties. In this paper, we describe a user interface framework that embraces a retrieval-centered interaction model to organize the user's activities in an information workspace.

The Information Grid (InfoGrid) framework provides a user interface design and a retrieval-centered interaction model for information access applications. In particular, InfoGrid is based on a paradigm of associative iterative retrieval of objects from loosely-structured or large swamps of information. The design was initially targeted at providing an easy-to-understand interface for managing documents stored as digital images in a distributed, multiuser, document database. Associative retrieval seemed particularly ap-

propriate in this case since the document database was designed to support a variety of organizational usages and thus had very little predefined structure. We built two implementations of this application in two different environments.

Subsequently, the user interface was refined to handle the needs of other retrieval applications, and we developed the InfoGrid object-oriented application framework. During the last ten years, object-oriented frameworks[8] have been employed quite successfully as a mechanism for capturing common infrastructure and policies in a reusable software base while still allowing enough variation to span a wide-reaching class of applications. Using the InfoGrid application framework, we have built a number of applications that conform to the InfoGrid user interface design including the application mentioned above; a browser for a database of lab members with pictures and biographies; an encyclopedia browser; and a personal/workgroup electronic file cabinet.

In this paper, we describe the InfoGrid user interface design and its interaction model, the object-oriented application framework for building InfoGrid applications, and a number of applications built using this object-oriented framework. We conclude the paper with a discussion of our experiences in using the InfoGrid design and application framework and a comparative critique of the retrieval-centered paradigm.

### INFOGRID DESIGN

In previous work, we have based our system design on the "Triple Agent Model" in which the system is thought of as three interacting agents: the User, a Dialogue Machine, and a Task Machine [22, 3, 17, 2]. The InfoGrid design extends this model by adding a Data Store to the Dialogue Machine as shown in Figure 1. This extension essentially separates the notion of a user task into an information access part and a set of actions. Henceforth, since our primary concern is with applications that provide document services, we shall

---

<sup>o</sup>Reprinted From: Proceedings of 1992 ACM Symposium on User Interface Software and Technology, Monterrey, CA, November 1992, ACM SIGGRAPH and SIGCHI

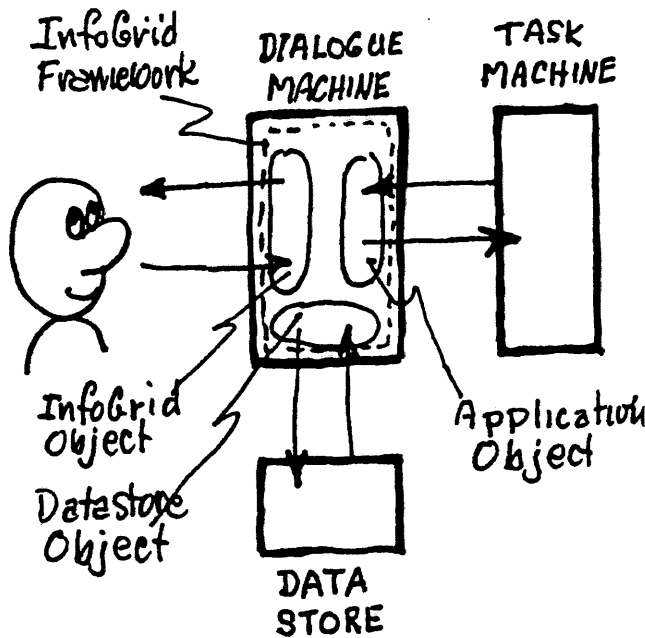


Figure 1: The InfoGrid extends the Triple Agent Model by adding a Data Store to the system containing User, Dialog Machine and Task Machine Agents. The InfoGrid Object-Oriented Application Framework resides on the Dialog Machine Agent and contains objects responsible for interfacing to the User, the Data Store, and the Task Machine.

discuss the retrieval of documents and the provision of document services. Thus:

$$\text{task} = \text{document services} + \text{information access}$$

In this context, the InfoGrid design can be understood in terms of (1) the user interface elements in the Dialogue Machine including a screen design and basic interface objects, (2) the information access mechanism supported by the Data Store, and (3) the document services supported by the Task Machine.

### User Interface Elements

The InfoGrid design relies on a number of well-understood elements including suggestive graphic design, a direct manipulation metaphor, and keyboard accelerators to achieve an easy-to-learn, comfortable-to-use interface. We have also explored using speech input and stylus gestures and supporting a range of contexts beyond standard keyboard and mouse-based window environments including copier interfaces, pen-based computers, and large display surfaces.

The InfoGrid screen is divided into tiled areas for interacting with the information access mechanism and

the document services provided by a particular application. These areas contain, primarily, two kinds of objects: (1) Property Sheets which display field-oriented, possibly editable, information and (2) InfoGrid Buttons. These two kinds of objects mostly address the user interface needs of the InfoGrid: iconic controls, document thumbnails, document descriptors, and entry boxes for requesting search.

InfoGrid Buttons are user-manipulable objects (see [18]) that react to stylus or mouse gestures, to drag and drop actions, and keyboard accelerators. Graphic effects (e.g. drop shadowing) indicate that Buttons are responsive to various user actions. Some gestures are assigned generic meanings for all Button objects: e.g. drawing a check mark selects a Button, scribbling over a Button deletes it, and grasping a Button moves it. Other Button gestures are assigned meanings based on the specific type of object that the Button represents: e.g. if a document Button is moved to the document area, then the document is retrieved and its thumbnail appears to expand into a full-sized page.

Figures 2 and 3 depict the screen design selected from several considered for implementation. The information access part of the interface consists of a search parameters area containing a property sheet that the user fills in to specify desired documents; a browser area where the system displays a visualization of the documents that matched the query; a holding area where the user can move interesting items from the visualization for subsequent use; and a search paths area that depicts the search history so that users can return to an earlier result. Once documents are retrieved, they can be operated on using tools from the control panel. Individual documents can be viewed in the document area, which contains an image or textual view of the document's content and a property sheet view of its description.

The layout and graphic design of the areas is intended to facilitate quick comprehension of their general purpose. The left side of the screen is devoted to retrieving, selecting, and operating on objects and the right side is devoted to browsing or editing a specific object. When appropriate, the full screen can be allocated to double-page viewing of the document. Keyboard accelerators allow expert users to move between the areas of the InfoGrid and select operations quickly.

### Information Access Mechanism

User interaction with an InfoGrid application is structured around retrieval as the top level user action. Retrieval is treated as an iterative process in which the user can go through several cycles critiquing query results as "relevance feedback" to subsequent queries [21]. This approach is similar to the "retrieval as con-

## CONCEPT #3 LAYOUT

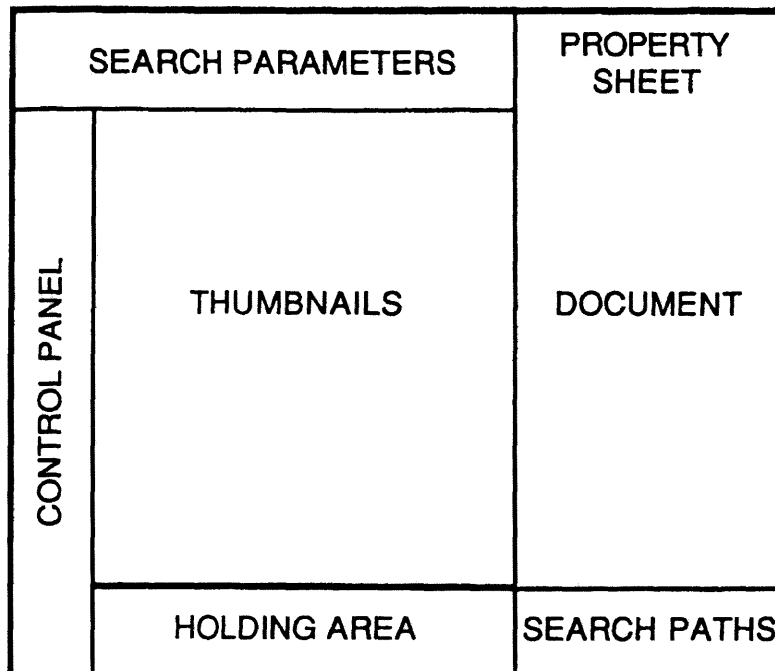


Figure 2: The layout of areas in the InfoGrid design.

versation” or “retrieval by reformulation” approach of Rabbit[26], Helgon[6], and WAIS[9]. Iteration and interaction are particularly important for situations in which the user doesn’t know what is available or what differentiations to make.

The user specifies an initial query by supplying values for various fields in the search parameter area. For example, as shown in Figure 3, a number of query fields are supplied by the user to seed the search. A search is then initiated by activating the search tool shown in the tool area using a pointer gesture, a keyboard accelerator, or a drag and drop action.

The results of the query are displayed in the results area as some kind of visualization. Currently, retrieved documents are shown as fixed-size thumbnail sketches that are generally laid out in a grid (hence, the name InfoGrid). Thumbnails are generated for a document by scaling the first page down to the required size. Documents can be moved to a holding area for subsequent use.

Retrieved documents can be used to seed subsequent search cycles. This is accomplished by selecting documents (by using the “checking” gesture) from the results or holding areas or portions of a document shown in the document area and invoking another search. For example, a number of documents can be checked and a “proximity” search can be invoked to find documents similar to the checked documents [20]. The search path area, shown in Figure 3, is meant to convey previous

search history and support returning to and refining previous queries.

### Document Services

InfoGrid provides access to various document services by allowing operations to be invoked on documents. Tool Buttons in the tool area represent services that can be invoked. The simplest service required of all InfoGrid applications is support for viewing or editing documents in the document area. Other common services that may apply across many applications include print, mail, and fax.

An InfoGrid application can install permanent tools for performing application-specific document services. Alternatively, the application can support retrieval of tools or tools with built-in arguments (e.g. print to a particular printer or fax to a particular number) using a tools search operation. Retrieved tool buttons can then be moved to the tool area. Thus, users can use the same mechanisms to tailor their workspaces as they use to operate on work objects.

### INFOGRID APPLICATION FRAMEWORK

Object-oriented application frameworks provides a powerful medium for implementing a software framework that divides responsibility between an architecture and a particular instantiation of that architecture. A number of object-oriented application frameworks (including MacApp[10] and Unidraw[25]) have successfully demonstrated this in various domains. More-

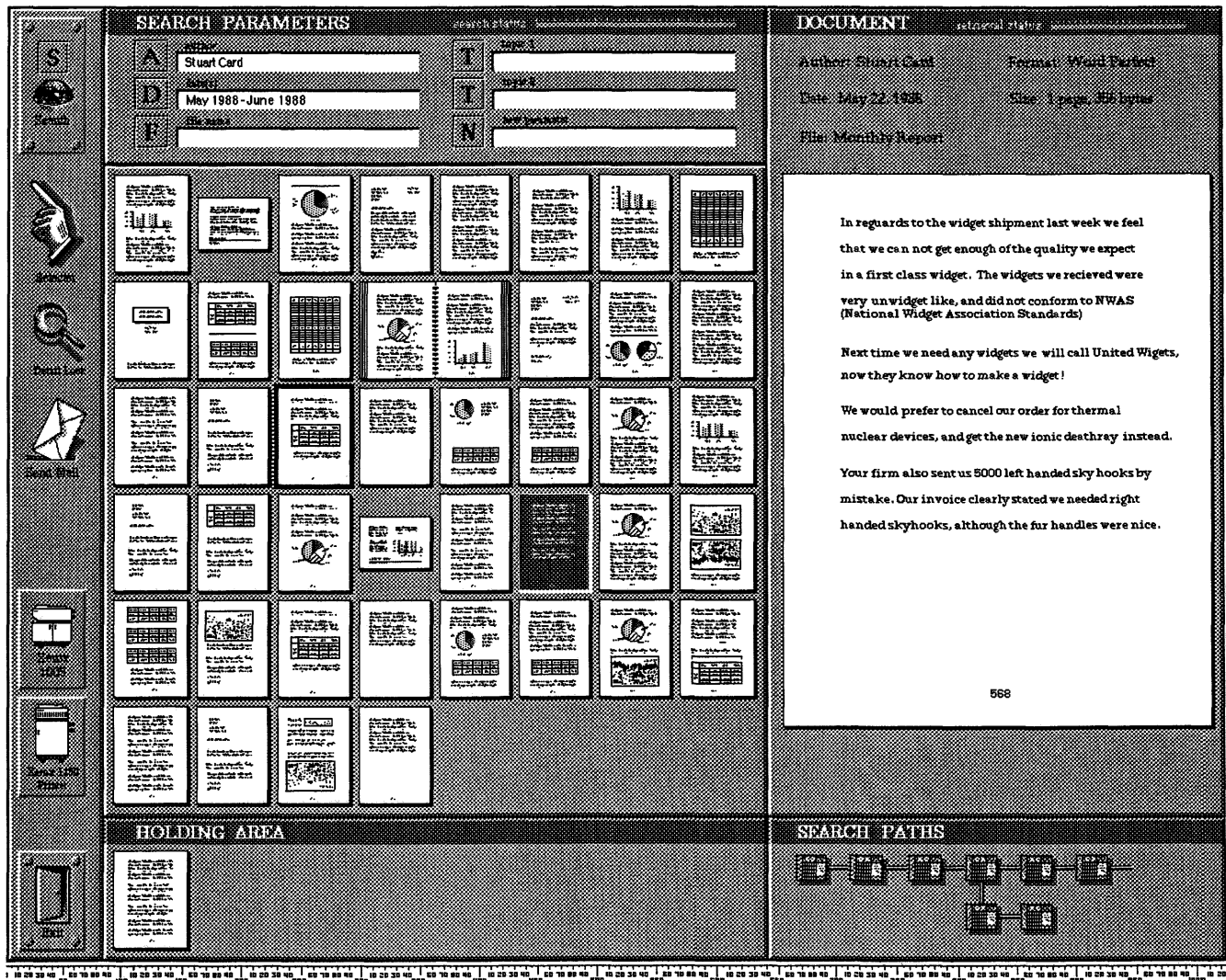


Figure 3: Design mockup of InfoGrid

over, there has been a long association between the object-oriented programming paradigm and the object-oriented user interface metaphor. It seemed only natural that we use object-oriented technology to build an application framework for the InfoGrid design.

The InfoGrid application framework described here is implemented using Common Lisp, the Common Lisp Object System, and the Common Lisp Interface Manager (a user interface programming interface)[24, 16]. However, the design would work equally well in other object-oriented languages, using other object-oriented user interface toolkits (e.g. Interviews[11]).

The InfoGrid application framework consists of a small set of classes that play various roles in the overall architecture and a few object-oriented protocols that define how requisite functionality and implementation is provided by the system. Each object-oriented protocol is a specification of a set of functions (or messages) that are invoked (sent) in some sequence and under certain conditions on (to) objects playing particular roles. A new InfoGrid application is developed by specializing various framework classes and providing particularized methods for protocol functions to implement application-specific behavior.

### Classes

The three major objects of the InfoGrid interaction model, as illustrated in Figure 1, are the InfoGrid, Application, and Document Store objects which interface to the User, the Document Services, and the Data Store Agents, respectively. The division between the InfoGrid, Application, and Document Store objects reflects the division between the user interface, the document services, and the information access mechanism. The InfoGrid framework provides classes for each of these objects as well as for other objects, most importantly Documents and Tools.

An application developer, at a minimum, must specialize the Application, Document Store, and Document classes and define about a dozen required methods to build an application. These three framework classes are all abstract classes, meaning they shouldn't be instantiated, but exist to provide common or default behavior for the benefit of concrete subclasses. Below, we describe the major responsibilities of the five major InfoGrid framework classes:

**InfoGrid** The intention is that an application developer need not specialize the InfoGrid class. It provides the top level interaction loop, and invokes various function on the other objects at appropriate times. In addition, it manages layout and control of InfoGrid areas and InfoGrid Buttons that represent document and tool objects; and it provides common user interface services including user feedback messages and search

progress indicators.

**Application** This is the major class that an application developer specializes to implement a specific application. All portions of the design that require application-specific implementation or that allow application-specific variation are defined as functions for which an application can provide methods. For example, an application is responsible for generating a set of tools; for a defining the fields of the search parameters area during initialization; and for providing functionality for browsing documents in the document area.

**Document Store** This class provides the interface to a particular information source (e.g. a database or a set of databases). In particular, a document store processes search parameters specified by the user, generates a query, and constructs document objects based on matches to the query. Though the design allows mixing and matching applications and document stores, currently, our InfoGrid applications consist of paired application and document store classes.

**Document** Document objects are produced by an application's Document Store object in response to a query. The Document class is an abstract class that an application developer may subclass multiple times to represent the various types of objects accessible in the Document Store. Each document is a handle to a chunk of information (i.e. object or document) in the document store. Each document class must provide functionality for accessing or otherwise deriving views of the document needed by the application. For example, a document class provides methods for generating thumbnail images and for retrieving the data needed by the application's browser component.

**Tool** This class represents application operations that can be invoked by the user usually on documents. Since retrieval is the central act in the InfoGrid design, the search tool is a required tool in every application. For simplicity, the tool class is designed as a concrete class that is parameterized with instance data. Thus, applications need not specialize this class, but rather can define new tools by providing different initialization data.

### Protocols

Object-oriented protocols define how various behaviors are provided by specifying a collection of functions that are invoked in some order, under some conditions, on objects playing particular roles. They provide opportunities to specialize behavior by allowing application developers to define new classes that support the roles. In the InfoGrid framework, an application developer defines subclasses of the application, document store,

and document classes and provides methods for functions in protocols that are defined on these roles. The following are the most important protocols:

**Initialization** During InfoGrid initialization, an initialize function is invoked on an application object. The application must provide a method for this function that constructs specific tools, initializes a document store, and generates user interface objects needed in the application-specific areas. In particular, it generates a search parameters property sheet, a query result visualization (usually a grid), and a document editor area using a common library of user interface functions defined on the infogrid class. This library helps maintain a consistent style across various areas (and various applications).

**Querying** When a search is invoked by the user (using the Search Tool Button), infogrid invokes a series of functions (initialize-query, perform-query, and finish-query) on the application. The initialize and finish functions provide optional hooks for allowing the applications to shadow standard feedback messages and to perform appropriate setup and cleanup. A method for the perform-query function must be provided by an application. The method should use the current search parameters (including selected documents or text) and the document store to obtain a new set of document objects.

**Editing** A user can select a document for more detailed browsing or editing using the document editing area. The application must provide a method for a browse function that initializes the document editing area using a particular document. The document editor interacts with the document object and perhaps its document store to access and operate on the document or information it represents.

**Tool Activation** When an application constructs its specific tools, it provides functions that are invoked when the tools are activated. This allows an application to provide access to various document services supported on its documents. In addition, the document service routine may periodically invoke functions on infogrid that queue progress feedback messages.

## INFOGRID APPLICATIONS

The InfoGrid application framework has been used to build four applications. These various applications have driven work on elaborating the user interface design and the application framework by exposing commonalities as well as differences between applications. In this section, we describe each of these applications, pointing out ways in which each application stressed

the InfoGrid design.

**Electronic File Cabinet** This application, shown in Figure 4, is an example of our original canonical problem domain. Documents are scanned and stored in a document store as a sequence of images and a set of attribute/value pairs. Users locate documents by querying on attribute values including scan-date, title, category, and keywords. In addition, we have incorporated optical character recognition technology to automatically extract a document's text and associate this with the document as a text field. Thus, the user may retrieve text documents based on contents. The document editor for this application allows viewing the document's images and text, and editing the documents attribute/value pairs. Tools for printing, faxing, and distributing to shared file cabinets are provided.

**Network Document Database** This application is similar to the electronic file cabinet in that it deals with documents that consist of a sequence of pages and an attribute/value description. The major differences are that (i) the document store is distributed across a network and contains many more documents from diverse sources and (ii) the document description schema isn't mandated by the system but rather by user conventions. Thus this application requires more careful design in certain aspects of its user interface. In particular, it is more important that feedback about status is managed carefully and that the user is given control over active processes (i.e. activated tools). Furthermore, this application relies on a more iterative approach toward search, where the user interacts with the document store to evolve more useful queries.

**Biography Database** This application, shown in Figure 5, allows retrieval of pictures and biographies of members of our research lab. Each person is treated as a document that can be retrieved by matching against name, research area, and/or words from their textual biography. Search can also be performed by selecting people in the results area and retrieving people that are most similar to them (based on a proximity measure between biographies). For example, by selecting a couple of linguists from an initial query, and initiating another search cycle, other linguists can be found. Small greyscale pictures are used as thumbnails. The document area displays a large picture of the person as well as their biography.

**Encyclopedia** This application, shown in Figure 6, allows a user to retrieve text entries from a six million word encyclopedia. This application (as well as others) use text retrieval technology developed by other members of our research lab, that incorporates support for

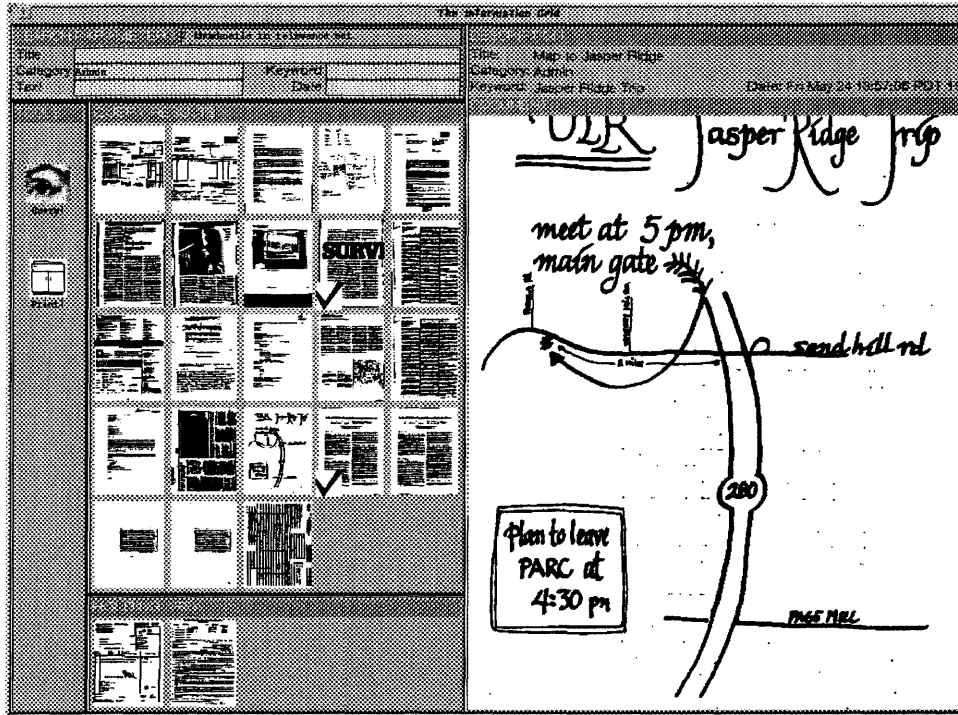


Figure 4: The Electronic File Cabinet application.

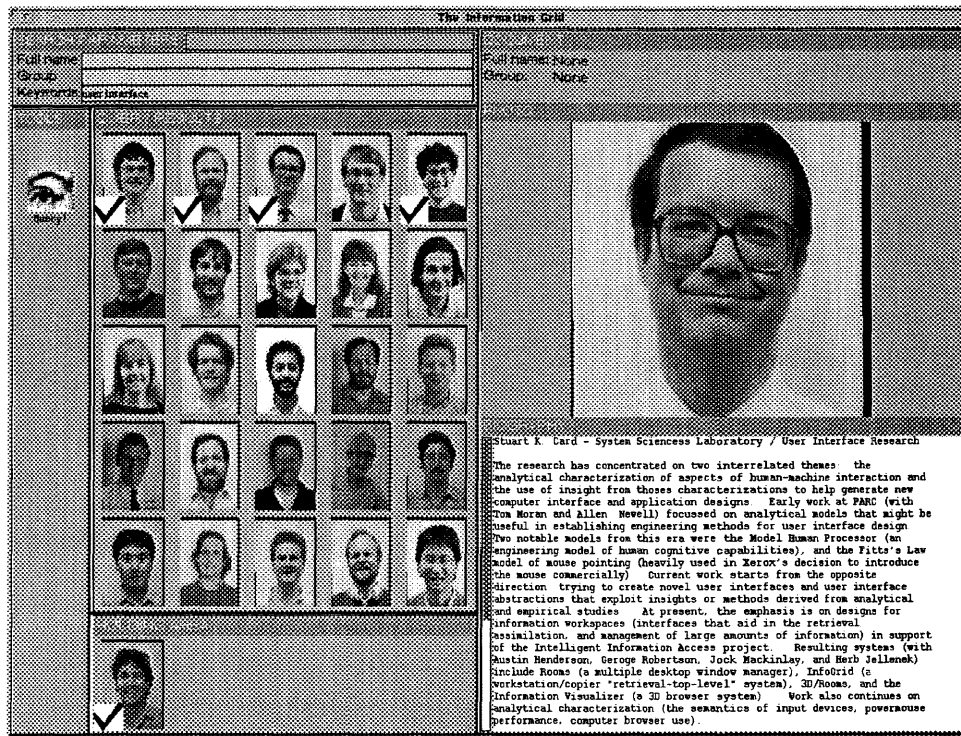


Figure 5: The Biography Retrieval application.

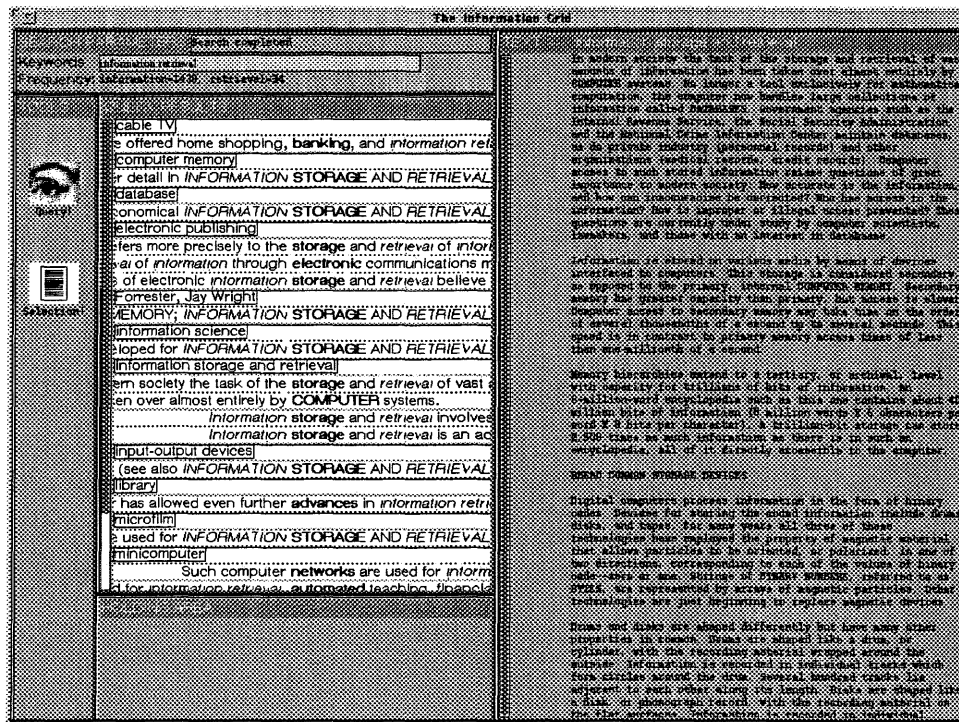


Figure 6: The Encyclopedia Browser application.

word stemming and other forms of linguistic analysis and for a variety of statistical search methods[5]. This application introduces text-based thumbnails and departs from the namesake grid layout to a list view. Its document area allows browsing of the information and selection of text to seed further search for other relevant entries. One issue raised by this application is that information from the store isn't always easily separated into distinct discrete entities. We chose to treat each encyclopedia entry as a document to conform to the model of our framework, but a more accurate story would have to deal with words, sentences, paragraphs, sections, and entries as meaningful units that coexist in a containment hierarchy.

## EXPERIENCE

The InfoGrid user interface design, despite its simplicity, has proven applicable to a variety of applications. Furthermore, the object-oriented application framework has made it possible to build a consistent family of retrieval-based applications quickly. In essence, the InfoGrid application framework is like a "HyperCard"[7] for retrieval applications or for applications with retrieval-centric interaction models. Like HyperCard, InfoGrid provides a narrow base of user interface objects and conceptual abstractions for building information interfaces.

The implementation of four applications using the

InfoGrid user interface design and object-oriented framework testifies to InfoGrid's broad applicability. Each of the applications has been tested or used by a number of users. Users are able to use InfoGrid with minimal initial instruction (approximately 5 minutes of training) and are able to use a second InfoGrid application without further guidance. In particular, users need to be familiarized with the iterative retrieval interaction loop, the purpose of the various InfoGrid panes and the gesture paradigm. Property sheets, buttons, and the drag-and-drop paradigm are familiar to most users. The Biography Database application was used in a walk-up situation during an "Open House" event. The Electronic File Cabinet application is being actively used by a number of users.

One problem with the current user interface design is that thumbnails are often insufficient as compressed representations of documents. They work well in the people browser, but are weak for distinguishing text documents or image documents within a class (e.g. purchase orders). We have begun to explore the use of thumbnails sketches that include a small amount of text. We have also incorporated text-based snippets (developed in [14]) in the encyclopedia application.

A major limitation of the current InfoGrid object-oriented framework implementation is that we rely on a scheduler outside of our control to schedule lightweight processes. Consequently, we do not have ef-

fective control over allocation of computation to particular activities (e.g. retrieval or other tool-triggered operations). This problem is further exacerbated in situations where network delays or other uncertainties can quickly make sloppy feedback frustrating (e.g. in the distributed document database application).

This very problem was one of the original motivations for the Cognitive Coprocessor architecture [17], a key element of the Information Visualizer system [2]. The Cognitive Coprocessor provides a tightly-governed animation loop that ensures a responsive user interface that does justice to the time constants of the user's perceptual and cognitive systems on the one hand, and of the various underlying computational processes on the other. The Information Visualizer's InfoGrid implementation utilizes this architecture to provide a more carefully regulated interaction between the user and the distributed document database.

A side effect of factoring the functionality into an object-oriented framework is that multiple InfoGrid applications can coexist in an address space at once. We defined a simple application swapping protocol that supports programmatically switching the InfoGrid from one application to another. However, we haven't designed a user interface for allowing the user to switch between applications.<sup>1</sup>

## RETRIEVAL-CENTERED WORKSPACES

In the location-centric/navigational paradigm, objects are conceptually located in places that closely mirror underlying structure, for example, storage models. Users either know where the objects they seek are stored or else navigate from place to place looking for the object they desire. For example, UNIX files are organized in a hierarchy and users either specify a complete pathname to obtain a file or else "walk around the file system" looking for the required file. Furthermore, most object-oriented or direct manipulation desktops are location-centric. Such desktops, including the Star[23] and the Macintosh[4], utilize analogies with the physical world to make the illusion of location and the process of navigation more concrete, but they do not depart significantly from the location-centric paradigm.

Computer-supported search provides an alternative mechanism for allowing the user to find objects. The `find` command in UNIX and a number of utilities provided for the IBM PC and Macintosh environments provide simple search-based mechanisms for automatically locating files. More recently, On Location and Lotus Magellan provide a retrieval front-end for accessing

files. The InfoGrid framework provides a generalized architecture for using the retrieval-centric paradigm.

A number of characteristics of an information workspace can influence the selection of one of these paradigms or the other. A location-centric workspace is reasonable when the information space is small or managed by a single user or when there is natural structure in the underlying information. For example, as pointed out in [2], information spaces with a hierarchical structure (e.g. Unix file system) or a linear structure (e.g. sequence of document creation dates) can be supported by appropriate navigable visualizations: Cone/Cam Trees[19] and Perspective Walls[12], respectively.

The location-centric paradigm, however, breaks down in a number of situations. First, a single location is not likely to serve the needs of a large numbers of users, especially if these include occasional users. Furthermore, if the information content of the workspace or the set of users changes frequently, then the coherence of the information space is likely to deteriorate rapidly. Finally, when the information set is relatively unstructured, structured visualizations can't easily be generated.

A retrieval loop is appropriate in these situations, since it allow users to utilize computation to locate objects associatively. Retrieving against multiple attributes, some user-specified, others machine-assigned, allows unbiased access along a number of dimensions as opposed to a single dimension (e.g. storage location). Thus, computation is used to locate objects (and possibly revisualize the space), not just to project a physically-consistent world.

Retrieval as the top level paradigm for a user interface is more broadly applicable than it might first appear. For example, most existing mail user interfaces support a location-centric view of mail folders containing messages. Users put messages in folders and find them by rummaging or searching through folders. An alternative is to allow a user to retrieve messages dynamically using both associated attributes (e.g. folder, keywords, new message status) and message header fields. These ideas have been pursued in Babar[15], the Information Lens[13], and database-oriented mail systems.

The InfoGrid interaction model can also be applied to the desktop. The desktop metaphor interaction model involves a single interaction loop between the user and the application residing in a window. The user switches between applications by selecting different windows. The Rooms interaction model[1] extended the desktop metaphor by positing that the user has a set of different tasks he or she is switching among, each of which may involve an ad hoc collection of ap-

---

<sup>1</sup>A user can, however, start multiple InfoGrids applications and use the window system to move between them.

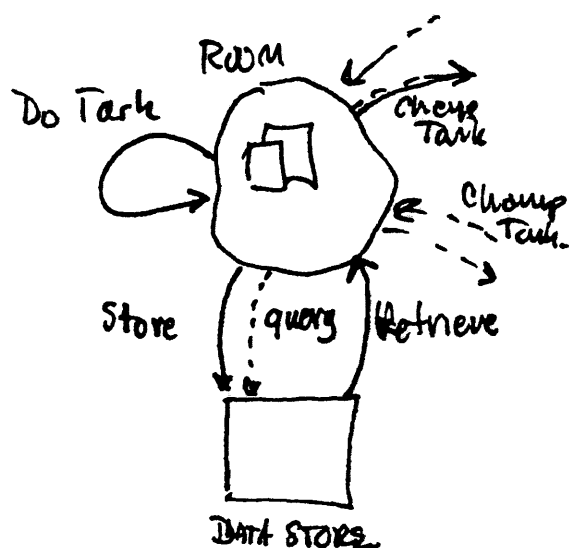


Figure 7: Interaction model for InfoGrid. This model refines the Rooms notion of user task by adding an information access component that supports storage/retrieval from a data store.

plications. The InfoGrid interaction model further divides the Rooms notion of task into an information access part and a set of actions.

The InfoGrid interaction model, depicted in Figure 7, contains three interaction loops between the user and the system. The first is the standard task cycle of actions on a document. The second is the iterative retrieval cycle of query/retrieval required to locate objects in the workspace. The final loop is the retrieval/storage loop. Information from documents that are retrieved is used to produce new documents that are stored back into the database.

The retrieval-centric paradigm provides an interesting and useful alternative to the location-centric paradigm of the desktop metaphor. Each of these paradigms has its particular strengths and weaknesses, and hence appropriate uses in building user interfaces. Ultimately, a hybrid of these paradigms may prove to be best for many systems.

## REFERENCES

- [1] S. K. Card and D. A. Henderson. A multiple, virtual-workspace interface to support user task switching. *ACM Transactions on Graphics*, 5(2), 1986.
- [2] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The information visualizer, an information workspace. In *Proceedings SIGCHI '91: Human Factors in Computing Systems*, pages 181–188. ACM, April 1991.

- [3] S.K. Card. Human factors and artificial intelligence. In A. Hancock and M.H. Chignell, editors, *Intelligent Interfaces: Theory, Research, and Design*, volume 1, pages 49–137. Elsevier Science Publishers, 1989.
- [4] Apple Computer. *Inside Macintosh*, volume 3. Addison-Wesley, Reading, MA, 1985.
- [5] D.R. Cutting, J. Pedersen, and P.-K. Halvorsen. An object-oriented architecture for text retrieval. In *Conference Proceedings of RIAO'91, Intelligent Text and Image Handling, Barcelona, Spain*, pages 285–298, April 1991.
- [6] G. Fischer and H. Nieper-Lemke. Helgon: Extending the retrieval reformulation paradigm. In *Proceedings of ACM CHI'89 Conference on Human Factors in Computing Systems*, pages 357–362, 1989.
- [7] Danny Goodman. *The Complete HyperCards 2.0 Handbook*. Bantam Books, New York, 1990.
- [8] R. Johnson and B. Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, June/July 1988.
- [9] B. Kahle, H. Morris, J. Goldman, T. Erickson, and J. Curran. Interfaces for distributed systems of information servers. Technical report, Thinking Machines, Inc, 1992.
- [10] K.Schmucker. Macapp: An application framework. *Byte*, pages 189–192, August 1986.
- [11] M. Linton, J. Vlissides, and P. Calder. Composing user interfaces with interviews. *IEEE Computer*, 22(2):8–22, Feb 1989.
- [12] J. D. Mackinlay, G. G. Robertson, and S. K. Card. The perspective wall: Detail and context smoothly integrated. In *Proceedings SIGCHI '91: Human Factors in Computing Systems*, pages 173–179. ACM, April 1991.
- [13] T.W. Malone, K.R. Grant, and F.A. Turbak. The information lens: An intelligent system for information sharing in organizations. In *Proceedings SIGCHI '86: Human Factors in Computing Systems*, pages 1–8. ACM, April 1986.
- [14] J. O. Pedersen, D. R. Cutting, and J. W. Tukey. Snippet search: a single phrase approach to text access. In *Proceedings of the 1991 Joint Statistical Meetings*. American Statistical Association, 1991. Also available as Xerox PARC technical report SSL-91-08.
- [15] S. Putz. Babar: An electronic mail database. Technical Report SSL-88-1, Xerox Palo Alto Research Center, 1988.
- [16] R. Rao, W. York, and D. Doughty. A guided tour of the common lisp interface manager. *Lisp Pointers*, 4(1), 1991.

- [17] G. G. Robertson, S. K. Card, and J. D. Mackinlay. The cognitive coprocessor architecture for interactive user interfaces. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 10–18. ACM Press, Nov 1989.
- [18] G. G. Robertson, D. A. Henderson, and S. K. Card. Buttons as first class objects on an x desktop. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*. ACM Press, Nov 1991.
- [19] G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proceedings SIGCHI '91: Human Factors in Computing Systems*, pages 189–194. ACM, April 1991.
- [20] G. Salton. *Automatic Text Processing*. Addison-Wesley, 1989.
- [21] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288–297, June 1990.
- [22] T.B. Sheridan. Supervisory control of remote manipulators, vehicles, and dynamic processes: Experiments in command and display aiding. In *Advances in Man-Machine Research*, volume 1, pages 49–137. JAI Press, 1984.
- [23] D. Smith, E. Harslem, C. Irby, and R. Kimball. The star user interface: an overview. In *Proceedings of the 1982 National Computer Conference*, pages 515–528, June 1982. Houston.
- [24] G.L. Steele. *Common Lisp: The Language (second edition)*. Digital Press, 1990.
- [25] J.M. Vlissides and M.A. Linton. Unidraw: A framework for building domain-specific graphical editors. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*. ACM Press, Nov 1989.
- [26] M.D. Williams, F.N. Tou, R. Fikes., A. Henderson, and T.W. Malone. Rabbit: Cognitive science in interface design. In *Proceedings of 4th Annual Conference of the Cognitive Science Society*, pages 82–85, 1982.

