CMU-ITC-091-106

The Alexandria Project

In support of an Information Environment

Michael L. Horowitz (mh11+@andrew.cmu.edu)
Fred Hansen
Michael McInerny
Thom Peters
Maria Wadlow

November 21, 1991

Information Technology Center
Carnegie Mellon University
Pittsburgh, PA 15213

Acknowledgments to **International Business Machines, Inc.**

The ITC Vision

At the ITC, we envision a future where scholars, artists, and other professionals use computers to create, manage, and communicate information as naturally as they use telephones.

The ITC Mission

The mission of the ITC is:

- to create a highly interactive, integrated, extensible computing environment in which people create, manage, and communicate multi-media information,

- to apply and test this environment in selected user communities at CMU and elsewhere,

- to gain international recognition as a center of excellence for distributed multi-media information systems, and

- to transfer our technology to our sponsor, IBM.

The Alexandria Project

In support of an Information Environment

Michael Horowitz
Fred Hansen
Michael McInerny
Thom Peters
Maria Wadlow

November 21, 1991

Introduction

With the development of advanced computer systems, we can now realistically think about providing access to exceptionally large amounts of information and to an environment that encourages the use and sharing of that information. Having such information on-line permits users to manage it in ways that far exceed simply providing access. Given the explosion of information availability currently taking place, users need tools that allow them to manage, manipulate, and share these flows of information.

Hypertext systems comprise an initial exploration into the issues concerning information structuring. Pure hypertext technology, however, cannot deal with the quantities of on-line information that will become available, even if a database is used as the underlying storage subsystem. More work is needed on the joint problems of access and management to have a meaningful impact on the way information is used.

The primary focus of the Alexandria research project at the Information Technology Center (ITC) of Carnegie Mellon University (CMU) is to investigate which tools computer users need in order to manage large amounts of on-line information over long periods of time. The characteristics that embody those tools include [Palay 90]:

1) *High performance* - The system must provide simple, fast access to large amounts of information from multiple, diverse sources.

2) *Flexible access* - The system must support a spectrum of access techniques from browsing (as in a hypertext system) to search (as in a database system).

3) *Personalized structuring* - The system must help the user productively manage information. In particular, it is not sufficient to provide access to information; the user should be able to impose a personalized structure, or organization, that facilitates later access to the information and aids the user to further his work. In addition, since such structure can become unwieldy, the user should be able to browse and search structure itself.[1]

4) *Structure evolution* - The system should help the user maintain and evolve the structure imposed on an information space. A user's view does not remain static; often as more information becomes available, the user will want to change the form of the structure, not just its content.

[1] "Structure" in this case is just additional information on top of other information; e.g. the table of contents of a book or the links in a hypertext document.

5) *Data type extensibility* - The system must accommodate a variety of digital media. Although we do not expect that the system initially will be able to recognize features from raster, graphic, audio, or video data, the design of the system must allow the management and access of such information.

6) *Collaboration* - The system should enable cooperative work within a community of users and encourage the exchange of ideas. Specifically, the system should make it easy for one user to view an information space using the structure built by another. To support such capabilities adequately, however, the system must ensure the privacy of unpublished data.

7) *Maintaining currency* - The system should handle changing information. Users often must keep pace with information sources that augment or replace previous information (e.g. news wires, electronic bulletin boards).

8) *Integration* - The system should be integrated fully into the user's computing environment. That is, all applications should be able to take advantage of the system's capabilities and users should be able to integrate data from any application into their personal information structure.

The Alexandria Vision

> *Users (i.e. scholars, artists, and other professionals) can, in a personalized fashion, manage and share all on-line sources of information seamlessly, without using separate, ad hoc applications for different sources and kinds of information.*

We desire an environment in which information creation, management, and communication occurs *naturally*. The Alexandria vision reflects our opinion that natural information management and sharing should occur *seamlessly* and *transparently*; that is, without boundaries between information sources or handling services. For example, it should be possible to relate information held in a library with information from netnews (or other electronic bulletin board system). Furthermore, it must be possible for each user to *tailor* the context in which information is managed and shared.

The Alexandria Mission

> *The mission of the Alexandria project is to enable interactive, personalized, long-term management, access, and sharing of large amounts of on-line information in an integrated and extensible fashion.*

Let's define some terms:

- By *large amounts*, we mean as much information as currently exists in the University Library; on the order of terabytes. Unless our system can deal efficiently with such large amounts of information, it might be difficult to convince potential users that it is worth working with Alexandria's technology.

- By *on-line information*, we mean information of arbitrary type from any reasonable on-line source, including over networks; information types of interest include complex models and new interpretations of data such as multi-media (e.g. audio, video).

- By *long-term*, we mean beyond a single session; that is, the state of a user's information management is persistent. In general, managing information is a task measured in years.

- By *integrated*, we mean that all handling of information occurs seamlessly and transparently regardless of its source or its nature. Furthermore, the system should be fully integrated into the user's computing environment. Thus, all Alexandria applications should be able to take advantage of provided capabilities, and the user should be able to integrate data from any application into their personal information structure.[2]

- By *personalized*, we mean that each user can choose a tailored configuration of options for dealing with the context of information management and sharing as well as the content of what is managed and shared. We believe that tailorability is a key capability for achieving user acceptance of our system; in particular, what is natural for one user may not be natural for another.

- By *extensible*, we mean that it should be possible to introduce new information types (e.g. multi-media), to incorporate new mechanisms for managing existing information types (e.g. document indexing), and to create new ways to structure, edit, and interact with information.

- By *management*, we mean that users should be able to organize and interrelate information from disparate sources in an effective fashion. *Effective* management means that the user can regain access to desired information at some later date and also that the user can synthesize new results from the interconnections of existing information. We believe this implies that users must be able to define information structures, impose those structures on existing information, and maintain and modify those relationships over long periods of time.

- By *access*, we mean that the system should support a spectrum of access techniques from browsing (as in a hypertext system) to search (as in a database system).

- By *sharing*, we mean to encourage collaboration among users so that shared information structures can be built and maintained. In addition, each user should be able to develop individualized views of common information and publish that "view", including any additional structure imposed on the information. Although the system should make it easy to share information and structure, it should also ensure privacy whenever requested. As noted in a previous paper [Palay 90], we have yet to determine the granularity of sharing that must be

[2] John Howard has coined the term "xeno-media" for this concept.

supported (i.e. interactive collaboration as proposed by Rubine and Keim [Rubine 91] vs. more normal, version concurrency control at the document level).

- Finally, by *interactive*, we mean that all activities in the system should support a user interface. In general, we hope to provide direct manipulation interfaces, but interpreted *little languages* may be required (e.g. a query language like OQL [Alashqur 89]).

Achieving any of these characteristics would contribute toward achieving the ITC's mission. Taken together, however, these characteristics still do not fully accomplish the ITC's mission. In particular, Alexandria's mission does not include exploring and solving the problems associated with creating, managing, and communicating specifically multi-media information (i.e. audio and video). Many of these problems we believe will be addressed by other ITC projects, and we intend to integrate the results of that work into Alexandria.

On the other hand, these characteristics, when taken together, do represent a large, coherent solution to many aspects of the problems the ITC wishes to address. In particular, Alexandria proposes to establish a framework within which most, if not all, information management applications (including those dealing with multi-media) can function and cooperate and to create an environment for users to manage such applications as well as their information. We call this framework and environment the *Information Habitat.*

Alexandria's Goals

We believe the design and implementation of the Information Habitat will demonstrate and fulfill Alexandria's mission. Doing so entails many subgoals, including providing the capabilities for personalized structuring and generalized access. One important subgoal that is not obvious involves enabling users to construct information management applications *interactively* by designing and implementing the tools needed to do so.

To demonstrate the usefulness of the Information Habitat, we feel it is necessary to design and implement a set of cooperating applications that satisfies the needs of a specific user class. To this end, we will build support for managing a computer scientist's research notebook. (Other "customer" applications are also being investigated.)

The next section describes the Information Habitat and motivates the different capabilities that it should provide. The following section delves into more detail about our reasoning for the need to support interactive application construction. Finally, the last section discusses the requirements on the system raised by one user application, the Research Notebook.

The Information Habitat

The ITC's mission describes an environment in which users deal with information. Alexandria's first goal is to provide an environment in which users can do more than simply access and edit information - they can also manage and share that information. Access is already provided by various database and file system interfaces. Applications provide the means to edit instances of information. Our goal is not just to generalize this interface and integrate applications, it is to extend functionality to enable personalized organization of the information and collaboration of such synthesis.

What currently represents a user's information space? Typically, a computer user's information space is kept in a file system and, sometimes, in various databases. A file system comprises an extremely poor interface to an information space. The (nearly) hierarchical name space provided by most file systems effectively acts as an index to the information contained in files. As an index, the file system provides only a single organization for browsing and very poor support for searching the information space. Furthermore, any meaningful semantic interconnections must be maintained by special-purpose programs

or by the user.

Of course, it would be better to present an interface that manages semantic content. Using hierarchical names is just one way to impose structure on a set of information entities. A fair amount of personalization is allowed; users can choose their own names and hierarchies. Simple names and hierarchies, however, are not appropriate structures for all kinds of information. As computers are used to manage more and different complex kinds of information, we must create an environment in which users can impose suitable structure on that information. In addition, users must be able to search that structure as well as browse.

Users will still wish to deal with different kinds of information in different ways. Users generally think about handling mail messages one way and managing source code and documentation another. The user interface for the Macintosh Finder[3] constitutes a leap for users in this regard. Information entities may be tagged not only with a name but also with a type, which can be represented by an icon. Selecting that information entity (i.e. double-clicking with the mouse) executes the correct application for that type of information. This interface has several implications. First, applications must get all other parameters from the user or from the operating environment. Second, each type of information may have only one application that might be invoked in this manner. Even so, the Macintosh interface still presents a file system: a hierarchical organization of names.

We propose a quantum leap from typical information organizations toward something which more closely reflects the way people think about their information. Instead of an environment in which the user can only browse a file name index for information of interest, the Alexandria *Information Habitat* will offer an interface in which users can: (1) browse or search for desired information as appropriate; (2) impose structure relevant to the information being managed (thus, enabling a high degree of personalized organization of that information); (3) view provided information and imposed structure transparently (i.e. without requiring one mode to view information and another to view structure); (4) share protected "views" (i.e. imposed structure) of some information with others; and (5) define and select multiple applications for handling information of each type.

To be integrated, our information environment must manage multiple sources and repositories for information (i.e. not just the file system). In particular, the facilities provided to users should apply equally well to such sources as the University Library, netnews, and several network databases (e.g. X.500 directory servers).

What, then, would be a typical scenario for a user operating in the Information Habitat? Initially, the user starts in a state in which he must specify information of interest. To do this, the user either names the information directly (using some index as the naming universe) or poses a query describing the characteristics of the desired information. In some cases, knowing how to ask the right question may be difficult, so it will be possible for the user to browse structure itself (thus discovering, for instance, that mail messages have a subject field). Once the information has been retrieved, an application appropriate to the type of the information can be chosen for interacting with and editing that information. The initial query interface and subsequent application interfaces constitute views on information buffers. On a system that supports a multiple-window environment, many such views can be active simultaneously. Each application can support the ability to impose additional structure on its information, to cut, paste, or refer to its information from other applications, and to publish its information for other users (including any imposed structure).

[3] Macintosh is a registered trademark of Apple, Inc.

To summarize: The Alexandria Information Habitat must support the following user activities:

- *imposing structure on information;*
- *interacting with imposed information structure, including browsing;*
- *posing queries based on imposed structure, and executing searches;*
- *selecting information domains for both browsing and searching;*
- *selecting how to interact with different types of information;*
- *publishing information for the benefit of other users;*
- *extending and integrating new mechanisms for accomplishing any of the above activities (including new handlers that manage typed information semantics - e.g. a compiler or symbolic math package).*

In particular, the Information Habitat design should have the following characteristics pertinent to the Alexandria Mission:

- *deal with large amounts of on-line information;*
- *enable interactive, long-term, personalized management and sharing of information; and*
- *provide an integrated approach to information management.*

The next section will outline how we wish to provide extensibility and enhance the interactive nature of the system.

Interactive Application Construction

> *[Aside: "Application" may not be the correct term. First, "application" brings to mind something heavy-weight that can stand on its own, whereas we desire an interactive activity that is integrated within an environment. Second, the term "application" has a technical meaning within certain environments. Perhaps better would be "little application" (in line with Bentley's "little language") or "information handler" or "information manager". One paper in the literature uses the term "mode" to refer to such mini-applications [Shan 90].]*

Recall that users will want to deal with differently typed information in different ways. Examples of such information handling applications include:

- mail and bboard handler
- source code browser (integrated with concurrency control and documentation maintenance)
- library search system
- document browser and editor
- other hypertext-like applications (e.g. help)
- various graph browsers (e.g. organization charts, zoological taxonomies)
- dictionary/thesaurus/encyclopedia browsers
- telephone directory service
- research notebook support
- calendar maintainer
- bibliography manager
- map maintainer (e.g. to provide driving directions)
- journalist's workbench

This list is certainly not exhaustive. Examining this list, however, leads to two conclusions. First, each application primarily involves simple, common information management: certain known structures are defined (e.g. card catalogue for the library system, table of contents for documents) and similar operations provided (e.g. search, follow link, zoom in for additional detail, add and remove entities). Very little of each application requires special-purpose knowledge (e.g. how to send or post a message in the mail handling system or how to edit rasters, drawings, or video).

Second, to be integrated within the Information Habitat, these applications must share several user capabilities in addition to the basic functionality of interacting with typed information, for example:

1) posing queries;
2) imposing structure;
3) transforming information structured one way into information structured another; and
4) publishing information and structure to other users.

The first two clearly derive from the goals of providing an integrated environment for information. The third results from the desire to interact with an information view that differs from any currently provided by any existing structure on that information. For example, a structured document browser and editor should be able to extract a document's table-of-contents from its stream representation. The fourth capability results from our desire to encourage collaboration, which motivated the choice of the project name Alexandria [Palay 90].

Because the list is not exhaustive, the Information Habitat must enable extensions to the set of applications integrated into the system. Because each application primarily involves common management capabilities, it should be possible to construct new applications from existing capabilities provided in the environment.

Significantly, the list of capabilities to be provided to users by these applications are also required by the implementors of those applications. The conclusion we draw is that since these capabilities should be provided interactively for users, these capabilities can also be provided interactively for implementors.

Thus, the Information Habitat must provide interactive "applications" to pose queries, select and impose structure, create and execute transformations, and create and invoke authorizations for publishing.

At this point, much of the implementor's work can be done interactively. How much of the rest can also be done interactively? To answer this question, we must first ask what capabilities are required in the "rest".

In addition to those described already, two new capabilities are required by application builders. First, an implementor must be able to define the structure of the information being manipulated.[4] This effectively involves defining data models. The second capability is to define how users edit and interact with instances of such structure schemas. This activity closely resembles dealing with a UIMS, except that, as noted above, the set of actions users may wish to execute within an information management application is mostly bounded.

Both of these activities can also be interactive. Certainly, making the latter interactive is the subject of considerable interest in the user interface research community. Since our domain (i.e. information management applications) is limited, it should be possible to take advantage of existing results. We will discuss how to make the structure schema definition phase interactive in more detail below.

Once these capabilities are provided within the Information Habitat, there is no reason not to provide them to end-users as means for achieving highly personalized extensibility.

Before we conclude this section, we should address the issues concerning the integration of third-party applications (e.g. `idraw`) and foreign data storage formats (e.g. `PostScript`[5]). Invoking third-party editing applications when appropriate should be fairly trivial; we have already posited that applications will be selected based on information type. All that is required is some mechanism to describe how to invoke the application (i.e. required environment variables and parameter specification). Of more interest is how

[4] Note: at this point, the reader might notice an inconsistency in the use of the term "structure". In this instance, we mean structure as schema, or something that defines the shape of information. Previously, we used "structure" to mean additional information a user associates with existing information. When context is not sufficient for disambiguation, we will say "structure schema" for the first meaning and "structure instance" for the second, since the additional information imposed by a user is in fact an instance of some structure schema.

[5] PostScript is a registered trademark of Adobe Systems, Inc.

to integrate Information Habitat capabilities into such applications. This will depend on the level of cooperation provided by the application. If source code is available, some programming effort may be required. On the other hand, if the application supplies a sufficiently powerful command language, less experienced users might be able to incorporate our capabilities.

Integrating foreign data formats into the environment can be accomplished at three levels. The most integration is achieved if that data format is recognized directly by the Information Habitat capabilities that we build ourselves. Clearly, the number of such formats must be limited. The second, most useful, level is accomplished by enabling the user to specify how to parse foreign data formats into the form managed by the environment and how to render an internal form back out to the foreign format. The issues of how users can specify such transformations interactively are discussed in more detail below. Finally, there may be data formats that simply cannot be analyzed. Objects of such formats can still be managed within the environment, they just can't be decomposed. In particular, it is these objects that will require the integration of third-party applications discussed above.

Thus, Alexandria's goals include supporting the interactive construction of information management applications by providing at least the following set of interactive capabilities within the Information Habitat:

- pose and execute queries;
- define, select, and impose structure;
- define and select how to interact with instances of structure;
- "run" an application (i.e. interact with selected information);
- define and execute transformations on structured information; and
- define and apply authorizations for publishing and collaboration.

Please note two things. First, these capabilities were motivated solely by the desire to have an Information Habitat in the first place. Second, once these capabilities are provided, the Information Habitat just becomes the driver, or framework, for these capabilities as well as other applications.

Finally, we believe that other capabilities exist that would also be advantageous, including:

- the ability to specify transformations that parse foreign data formats into a form useful within the Information Habitat and render modifications back out in the foreign data format;
- the ability to define and apply indexes to sets of information entities (e.g. documents) in order to enhance the performance of queries; and
- the ability to define and schedule agents that execute on behalf of users in the background, "running" applications (combining agents and transformations allows automatic structuring to occur, such as automatic classification; agents can also act as a "current awareness service" for information sources that are constantly being updated).

In time, we may find the need for others as well.

To summarize: Supporting application construction in the Information Habitat should be as interactive as possible since the activities involved in building applications greatly overlap with the capabilities we wish to provide to users for tailorability and extensibility.

A look ahead: To achieve the various characteristics we desire, we hypothesize that the following support technologies will be necessary --

- *"long-term" management requires persistence (either via file system, databases, or both);*
- *search access requires the ability to pose and execute queries and to follow direct links;*
- *sharing requires the ability to define authorizations and some form of concurrency control;*
- *personalized management requires the ability to define and impose structure;*
- *integration requires control over information sources, interoperability, distribution support, and defined communication protocols between applications;*

- *tailoring requires user access to all building activities and interactive editing of the properties of the insets used to interact with information;*
- *extensibility requires at least the activities described above; and*
- *"interactive" management of information requires either direct manipulation interfaces or simple "little languages".*

The Research Notebook

The concepts embodied by Alexandria will be tested by specifying and implementing a set of cooperating information management applications within the Information Habitat to satisfy the needs of a specific group of users. This goal is important in demonstrating not only that one's information needs can be met within a single, integrated environment, but also that the applications satisfying those needs can be built interactively.

Thus far, computer technology has provided us with the means by which we can accomplish some tasks more rapidly than before. Using current computer applications, authors, for example, can produce written texts more quickly. But computers still do little to help authors understand the problem domain, apply that understanding to the task at hand, produce better solutions, or find solutions where they could not find them before. In short, current information technologies remain unsatisfactory for significant parts of the writing process.

Computers provide a powerful environment in which researchers, artists, designers, writers and other professionals can do their work. In fiction writing, for example, the computer provides tools which support editing, formatting and even spelling. However, it is the author's responsibility to keep track of the intended audience, the plot threads, the character development and the sequencing of events or other information that is needed to write the piece. The computer is helpful only after most of the difficult work is finished. Outline processors are a step in the right direction, but they still provide little help in structuring and managing ideas. Similarly, large portions of the graphic design process are currently unsupported in information systems. A graphic designer must manage a variety of ideas over a period of time. Those ideas are often represented in different ways (i.e. sketches on paper, verbal descriptions, images, etc.) and organizing them is often problematic.

What is needed for the research process is a flexible method of organizing, accessing, linking, and tracking information, goals and ideas. An ideal computer environment would be one which supports these critical, yet often overlooked, tasks. Imagine, for instance, computer-based tools to support a composer's creation of a sonata, including sketches for its design. Such tools might help monitor the changing shape of the composition, manage the generation, storage, and evaluation of alternate phrasings, and review all or parts of the sonata. Or, alternatively, a graphic designer might find computer tools useful for managing visual concepts and ideas, as well as for tracking the progress of the emerging design.

Our aim is to develop information technologies to support the research process more effectively. Research might apply to the design of buildings, machines, or other real-world objects, the composition of musical scores or written essays, the production of dance or video presentations, or the construction of computer applications or tools. The technologies we will be exploring can include new devices (e.g., videodisks) as well as new interfaces derived from a better understanding of the research process itself. Our current proposal for achieving our goal is to build a *Research Notebook* for computer scientists.

A typical researcher has access to large amounts of (potentially) on-line information: the library card catalogue, journal abstracts, journal articles, videotaped lectures, electronic mail and bulletin boards, newswires, etc. However, easy access is meaningless if the researcher is unable to find the relevant information when it is needed or relate it to other information effectively. A researcher may browse through a collection of information, either searching for information on a particular topic or looking for new ideas or perspectives on that topic. This search may last ten minutes or ten months. The researcher may find an answer and move on, or may wish to continue browsing at a later time from a different starting

point. The researcher may also wish to record the path that this search takes, leaving a trail which may be followed at a later date, or mark bits of information of particular interest, so that they can easily be found again.

It is vital for researchers to keep up-to-date on topics relevant to their field of expertise. So much information is being published in so many areas that this is becoming an impossible task. Hours may be spent sifting through piles of uninteresting data in order to find one or two nuggets of useful information. An alternative might be to narrow one's specialization drastically, investigating only those areas likely to produce pertinent information. However, by limiting the scope of examination, a researcher may overlook potentially crucial data or relationships.

In addition to finding information and keeping abreast of the latest activity in a particular field, researchers also need to keep notes recording current ideas, trains of thought, relevant conversations, relationships between pieces of information, etc. These notes indicate the state of the research, a snapshot of the work in progress. They should be connected to germane reference materials as well as to other information created by the researcher. In addition, a researcher may work on several projects simultaneously. A method is therefore required for recording progress so that no ideas are lost while switching from one project to the next. Or, a researcher may collaborate with colleagues and wish to make public any findings or new ideas. Finally, the researcher may need to organize ideas and their relationships, create theories, models, and explanations for observed behaviors, and test those theories empirically.

Having access to compilations of experimental results, theory revisions, related work, comments, and notes would aid in the research process, but more importantly, the researcher needs tools that understand the interrelationships inherent in that data. In addition to such research tools, several support applications, such as a management application for annotated bibliographies, are also desirable.

To summarize: The Research Notebook will need to support not only management of the different kinds of information available to the researcher, but also the synthesis that goes on during active research and the transitions that occur when the researcher switches between projects.

The Research Notebook must support the following user activities:

- *accessing an integrated multi-media environment;*
- *managing multiple threads of activity;*
- *structuring ideas;*
- *linking information of any type (text, rasters, hierarchies, etc.), both public and private;*
- *publishing information, including links, data structures, maps and audit trails;*
- *tracking revisions in personal data and updates in public data;*
- *searching and browsing in personal databases and large-scale public databases;*
- *specifying search criteria for ongoing searches;*
- *keeping audit trails on searching and browsing which persist in the long term;*
- *structuring notes, audit trails, links, etc. into multiple projects and switching between project contexts;*
- *managing more mundane aspects of the research process, such as bibliographies, research papers, talks, etc.*

System Outline

In this section, we discuss in more detail the interactive activities Alexandria will support. It describes the capabilities that end-users and application builders might expect from each activity.

We can divide users into several classes:

> *Novice users* will tend to use only those facilities that are provided directly by the application and will not attempt to extend those facilities, even by defining macros.
>
> *Knowledgeable users* are aware of the underlying models of the application and feel comfortable using the extension mechanisms that don't involve complex programming (i.e. programming that requires package-level extensions - defining data types and modules).
>
> *Expert users* are knowledgeable users that are able to write fairly large programs in order to extend the application. To some degree, these users are also able to make use of trapdoors to the language used to implement the application.
>
> *Wizards* understand how the application was built and can write programs that take advantage of that knowledge.

Below, we annotate some actions with the level of user sophistication we feel will be necessary to perform those actions. All other actions should require only novice sophistication.

We also distinguish between *end-users* and *application builders*. End-users are those users who interact with Information Habitat applications and the activities below that are part of those applications (e.g. specifying and executing queries). On the other hand, application builders use all of the activities described below to construct new information management applications. Generally, application builders will need to be more sophisticated than novices, although novices may build simple applications. End-users, of course, can be of any level of sophistication.

The following is organized according to logical activities without any consideration as to how a user interface to such capabilities might appear. In particular, actions of several activities may appear within the same user interface (e.g. selecting a structure to impose on information may be integrated with the activity of viewing information).

Similarly, a user might start in any of several activities, regardless of the order presented below. Each activity is annotated with the level of user sophistication expected for using that activity as an entry point. (*Note that one cannot start viewing information or imposing structure without somehow "retrieving" the information to be managed.*) In reading the following, it might be helpful to refer to Figure 1.

Please note that all actions can be performed only by users authorized to do so. For instance, almost everyone should be able to specify and execute a query, but some information domains may be off-limits to some users. Control over authorizations, of course, also requires appropriate authorizations!

Toolkit Outline

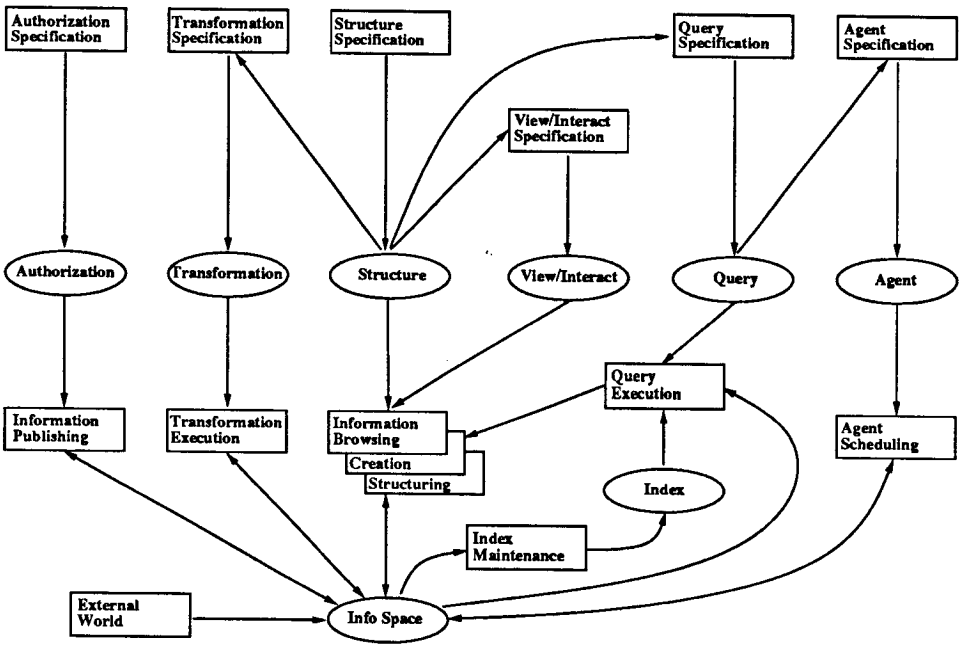

Figure 1: Outline of Information Habitat Capabilities

(0) Create Information [novice entry point]

This activity is not significantly different from the view, edit, and interact activity (described in the next subsection); creating information, however, can be an entry point whereas viewing, editing, and interacting with existing information cannot. Thus, although we distinguish the two for exposition, these two activities should be treated as one. In particular, any references below to the view, edit, and interact activity implicitly refer to the activity of information creation as well.

The following action is available in this activity:

(a) Create new information
- This action involves selecting a structure from a library of known structure schemas or creating a new schema (see activity (3) below). Instead of imposing that structure on existing information (as in action (g) of the view, edit, interact activity), the user creates new information by filling in data of the appropriate types (using the editing capabilities of action (a) of the view, edit, and interact activity below). Creating new information will involve an application switch. The structure schema selected restricts the set of application interfaces from which the user may choose.

(1) View, Edit, and Interact with Information [not an entry point]

In this activity of the Information Habitat, all actions manipulate information through some structure. In particular, a user imposes new structure on existing structure (i.e. the structure currently being viewed). Similarly, any processing of the information (e.g. queries) also operates on structure. Simple data (e.g. plain text) is considered the degenerate case of structured information.

The following actions are available in this activity:

(a) View, edit, and process information of a specific type
- Each type will present view and edit actions appropriate to it, including scrolling, playing (for continuous-time media), zooming, abstracting, searching (e.g. string search in text), analyzing, transforming (e.g. scaling, moving, rotating), and other ways to change its value.[6]
- Editing may involve some interaction with the database to "start" a transaction. Nested transactions, then, can be used for checkpointing.
- The specification of some information processing may require the skills of either a knowledgeable, expert, or wizard user. *In particular, there may need to be a trapdoor to allow access to a general-purpose language for specifying processing routines.*

(b) Save changes
- This action would terminate the current editing transaction and cause new versions of information modified during the session to be stored in an appropriate database.

(c) View and edit the structure of information

[6] For this, we expect to take advantage of the underlying user interface toolkit technology and third-party applications, as well as the results of research into continuous time media authoring currently going on at the ITC.

- Although many different structures are possible, most will share view and edit actions similar to above. The difference is that these operations will apply to the structure, and, as such, must be supported explicitly by Alexandria. The insets that provide this capability are called Information Interactors, or *InfoActors*.[7]

(d) Search and browse

- Search involves the actions associated with posing and executing queries (see (2) below). Browse typically involves scrolling and abstracting structure, and following links. Of course, what appears to be browsing to the user may in fact be the execution of an application-defined query.

(e) Change view/interaction [knowledgeable users]

- Each application understands information structured in a certain way. There may in fact be several applications that understand information possessing such structure. Users may use this capability to change their view of the information. When selecting the application interface with which to edit and interact with information, only those choices appropriate to the structure of that information should be available.

(f) Establish authorizations

- See activity (6) below.

(g) Select and impose structure

- See activity (3) below.

[7] The term InfoActors was coined by Thom Peters.

(2) Pose and Execute Queries [novice user entry point]

As mentioned above, no implication is made concerning user interfaces. Although we discuss this activity as separate from the previous, for instance, the activity of viewing, editing, and interacting may allow queries to be specified and executed within its user interface.

The following actions are available in this activity:

(a) Specify and edit queries

- Many different models exist for posing queries (e.g. keyword, predicate, query-by-example, form fill-in). The Information Habitat will provide a library of such models for use both by end-users and application builders. In addition, the architecture will support the integration of new models as they become available. We do not intend to provide an interactive interface for creating totally new query models; composition will be possible through the normal process of application building.
- Some models may be so complex that non-novice user sophistication may be required to specify queries in those models.
- Some query models may also accommodate user-defined predicates; *a trapdoor to a programming language may be necessary.*

(b) Save query [knowledgeable users]

- It will be possible to build up libraries of useful queries. Note that a library is simply some form of collection structure on a set of objects, in this case, queries.

(c) Browse and search query libraries [knowledgeable users]

- This action, of course, is just applying activity (1) above to query libraries. Once a query is selected, it can be edited using the capabilities presented by action (a) above.

(d) Execute query

- The execution of a query requires that the user (or application) specify the domain from which information is desired, the preferred accuracy of the results (i.e. recall and/or precision), and perhaps the indexes to be used for improving execution speed.

(e) Reformulate query

- The execution of a query results in a set of targets. A subset of this set can be judged by the user as to the relevance of the item to the user's information need. The system can then reformulate a new query, based on the user's feedback, and generate a new set. This process can be iterated.

(3) Select and Impose Structure [not an entry point]

Again, no implication is made concerning user interfaces. Although we discuss this activity as separate from the previous, for instance, the activity of view/interaction may allow structure imposition to be specified and applied within its user interface.

The following actions are available in this activity:

(a) Browse and search structure libraries

- Novice users will need a set of common structures from which to choose; they will not wish to define their own structure abstractions except in the most primitive circumstances. Examples of organizations that should be provided include annotations, bookmarks, cross-references, and standard application entities such as mail messages, card catalogues, dictionaries, calendars, telephone directories, etc. Primitive structure definition could involve, for instance, lists of records, where the user gets to specify the fields of each record.
- As in the case of query libraries, this capability is just applying activity (1) above to structure libraries.

(b) Instantiate structure schema

- Instantiation can be done directly by the user (e.g. during creating information; see action (h) of activity (1) above) or automatically through transformations (specified and executed via activity (8) below or applied by the execution of an agent; see activity (7) below).

(c) Evolve structured information [knowledgeable user]

- As time goes by, a user's view of parts of his information space may change (e.g. as when a researcher's focus moves to an area in which she only had a passing interest previously). In this case, the structure imposed on that information must change. Although the user can make these changes by hand, the action must provide some support for automating this task (perhaps with the use of transformations).
- This capability also has some implications concerning information versioning. For example, it should be possible to monitor changes to structure (i.e. changing structure constitutes an event that might invoke an agent). We have not explored these implications yet.

(4) Create New Structure Types [knowledgeable user entry point]

Since this activity is just another information management application, we expect it will have its own user interface.[8] In this activity, users, whether end-users or application builders, will define new abstractions for imposing structure. Application builders may do so as part of building up an application to manage instances of such abstractions. End-users, on the other hand, may do so to generalize and simplify the action of imposing structure. For example, if the system-provided structure library included annotation and cross-references as known structure abstractions, and the user wants to create annotated cross-references, then he would use this activity to compose the two into a new abstraction.

The following actions are available in this activity:

(a) Specify and edit structure schemas
- Specifying a new schema consists of creating new classes of information and listing their attributes and interrelationships. A model that users accept for defining such schemas (probably based on the entity-relationship model [Chen 76, Carlson 89, Su 89]) must be chosen. Attribute values and relationship roles may involve existing schemas; composing two or more existing schemas into one should be a fairly common occurrence. Thus, browsing and searching libraries of existing structure schemas (see action (a) of activity (3) above) must be accessible from this activity.
- Editing existing structure schemas (i.e. adding, deleting, and modifying attributes, changing relationships) requires evolution support. See also the related action of evolving actual structure (action (c) of activity (3) above).

(b) Save structure schemas
- Effectively, this is how structure schema libraries will be built. As before, a library is simply some form of collection structure on a set of objects; in this case, structure schemas themselves.

(c) Impose sorting criteria on structure schema
- Much of structure involves collections of subentities. Users generally prefer to view a collection in some order that is logical according to the information contained by the collection. To this end, users will be able to define sorting criteria on collections as part of defining structure schemas. Some known sorting criteria and ways to compose them will be made available to the user:
  * string sorts (case insensitive, ignore punctuation, ignore leading/trailing padding)
  * number sorts (float vs. integer, date/time)
  * table-driven (i.e. ordering a finite set of entities by hand) [expert users]
  * user-defined functions (requires programming; *a trapdoor is probably required*) [wizards]
  * multiple key (i.e. radix) [expert users]
  * multiple dimension (e.g. maps) [expert users]

[8] Although this activity is crucial for defining new applications, it is a candidate for being re-implemented as part of a boot-strapping phase (i.e. using the Information Habitat to re-implement this "application").

(5) Create New Information Application [expert user entry point]

This activity is the User Interface Management System (UIMS) for the Information Habitat. However, it must be more. As in a typical UIMS, an application builder can specify the appearance and interaction style of the application's user interface. Facilities must also be provided, though, that enable the application builder to specify how to filter structured information from a data source into the user interface and how to return modified information to the database.

Building user interfaces, then, will involve laying out Information Interactors (or InfoActors) and specifying the styles peculiar to each (e.g. the shape of nodes in a graph editor, the text font for entries in a title list). In addition to styles, each interactor will present a set of events generated as a result of end-user actions (e.g. item selection). The application builder can choose an information-based "semantic" action from a known set to be executed whenever that event occurs. (Known actions will include such things as execute a query, follow a link, create a new item, etc.) To incorporate non-information-based semantics, the builder will have to use a trapdoor into a full-fledged programming language; examples include interfacing to the mail delivery subsystem or invoking a compiler (for a source code browser). As we argued earlier, we believe that such applications primarily use only simple, information-based actions. One result of our work will be to determine whether this hypothesis is valid.

The other major component of building an application interface involves specifying how each interactor gets filled in from the information source and how to send editing changes back. For example, the table of contents of a book may be defined as a hierarchy and the user might choose a tree browser to interact with it. The application builder must somehow indicate that the node labels should be taken from the book's section titles. Furthermore, when the structure changes (e.g. sections are added or moved), the modifications must be committed to the database. See action (c) below.

The set of InfoActors available for composing application user interfaces will be limited. Interactive construction of such insets appears to be an extremely hard problem. Thus, we plan to provide a set of interactors that will represent a reasonable spectrum of interaction possibilities for several common information structures (e.g. sets, sequences, hierarchies, lattices, general graphs, lists of numbers [i.e. bar chart]). As in the case for data types (i.e. raster, audio, video), we hope to take advantage of the underlying user interface toolkit's extensibility to integrate new interactors into the system. These interactors, however, will have to be written in that toolkit's general-purpose implementation language (*another trapdoor*).

Since it is not expected that this activity will be part of some other activity, it probably should have its own user interface. It is probable, however, that it will not be possible to re-implement this interface as part of a bootstrapping phase.

The following actions are available in this activity:

(a) Specify and edit application interface layouts

- This action requires the services of a layout controller, much like that described by Cardelli [Cardelli 87]. The properties we desire include:
  * the ability to design screen layouts containing InfoActors;
  * the ability to impose layout constraints;
  * interactive editing of each InfoActor's styles; and
  * interpreting the resulting specification as an inset (instead of needing to compile).

(b) Specify and edit application interface interaction

- This action might be integrated with the layout controller. Using this action, a user can associate application actions with the events generated by end-users. Each InfoActor defines its own set of events (e.g. item selection). The application actions will fall into three classes: standard, information-based actions (described above); arbitrary actions defined by a wizard using a *trapdoor* into some general-purpose programming language; and simple macros of existing actions.
- We will attempt to provide a set of information-based actions that is as complete as possible. This is essential if we are to prove that most of the user interface of such applications can be built interactively.
- Although we do not go into such detail here, the ability (of wizards) to add to the provided set of actions implies the need for maintaining libraries for those actions.

(c) Specify and edit application interface I/O transformations

- In choosing the set of InfoActors for a screen layout (i.e. application user interface), the application builder must also describe the type of information that the layout is intended to manage and then how each InfoActor gets "filled" from instances of that information type. (By type of information, of course, we mean the structure schema of the information.)
- In addition to specifying how to extract the data needed by each InfoActor from the information source, the application builder must also specify how to package up any modifications that end-users make that must be returned to a database.
- There will be a controller in the system that will know how to take structured information and these layout transformation specifications and create the actual layouts with which end-users will interact.

(d) Save an application interface

- It will be possible to build up libraries of application interfaces. Note that a library is simply some form of collection structure on a set of objects, in this case, interface specifications.

(e) Browse and search libraries of application interfaces

- This action, of course, is just applying activity (1) above to libraries of interface specifications. Once an interface is selected, it can be edited using the capabilities presented by actions (a), (b), and (c) above.

(6) Manage Authorizations, Publish Information [novice user entry point]

An authorization indicates whether or not a specific user can execute a specific operation on a specific Information Habitat entity. Authorizations help maintain privacy and prevent unsanctioned updates. Storing each authorization explicitly is clearly not feasible. Most systems derive authorizations implicitly through the use of capabilities or access lists, neither of which are very flexible.

The Information Habitat will incorporate a design described by Rabitti, Woelk, and Kim [Rabitti 88]. As described in more detail in a companion paper on the architecture of Alexandria's implementation [Horowitz 92], three lattices are made available to the user so that most authorizations can be determined implicitly. The three lattices represent user groups, operation categories, and entity compositions. By establishing explicit authorizations involving nodes high in each lattice, all combinations of descendant nodes become authorized implicitly.

Thus, the activities described in this section involve defining the nodes of each lattice and managing the activation and suspension of specific, explicit authorizations. For example, we might define a user group for Alexandria group members, the category of all authorization operations, and the set of entities representing authorizations. With one positive authorization, all Alexandria group members can be authorized to manage any authorizations. Then, with one negative authorization, we can forbid a given user from actually removing anyone's authorizations.

This activity may appear as a subactivity of the others. Also, managing authorizations is clearly another information management application, and is thus a candidate for re-implementation within the system during a bootstrapping phase.

The following actions are available in this activity:

(a) Create and manage user groups, operation categories, and object compositions

- Managing each of the three lattices involves creating new nodes and placing those nodes within the lattice (node deletion can be a problem if there are existing explicit authorizations that refer to that node). Placing nodes involves specifying which other nodes are parents and which are children. Thus, any explicit authorization involving the new node implicitly authorizes all descendant nodes, unless overridden by another explicit authorization. Similarly, the nearest explicit authorization of an ancestor implicitly authorizes the new node. An algorithm for choosing between two conflicting explicit authorizations for ancestors at the same distance must be developed and published.

(b) Create and delete explicit authorizations

- Creating an explicit authorization means specifying a triple consisting of a node from each lattice (user group, operation category, and object composition) and registering that triple as comprising a positive or negative explicit authorization (a negative explicit authorization overrides a positive implicit authorization; e.g. everyone but Bill can read a given document).

(c) Activate and suspend explicit authorizations

- Instead of deleting an explicit authorization, it might be useful to just suspend its effect, so that the user need not re-construct it.[9]

[9] Maybe not; establishing an authorization looks to be a very easy operation.

(d) Query authorizations (explicit and implicit)

- Given the three lattices, it may not be clear to the user what the algorithm will decide is the appropriate implicit authorization for a triple that has no explicit authorization. Thus, it should be possible to ask what authorization (i.e. positive or negative) is associated with a given triple.

(7) Specify and Schedule Agents [knowledgeable user entry point]

Specifying agents involves three tasks: identifying the information of interest; specifying when the agent should execute; and describing what to do with the information of interest. The first task can be handled by specifying a query and its execution context. The second task involves the definition of various kinds of events and allowing the user to schedule the execution of agents according to when those events occur. Finally, the third task requires some form of programming. If we hope to avoid the need for a general-purpose language, we must identify the capabilities we wish to give the user and design an appropriate "little language" [Bentley 86].

Agents can be used to monitor dynamically changing information (either additions or modifications) and to automate the imposition of structure (e.g. classification, indexing). Thus, the capabilities we expect the user might require in our "little language" should include a programmatic interface of all the actions being described in this report. This implies that the language must also allow users to manipulate instances of Alexandria entities (e.g. structure schemas, queries, transformations, authorizations, agent scheduler events, structure instances, and agents themselves).

We expect the agent scheduler to understand at least the following event classes:

- time (e.g. run this agent every Monday at 3:00 pm);
- the addition of new information; this enables the automation of the "current awareness service" typically provided by libraries to inform library patrons of new books or articles that may specifically be of interest to them;
- the modification of existing information; this capability allows the system to simulate the database "trigger" concept; and
- other Information Habitat events of interest, including any changes to authorizations (i.e. for publishing information) or when another user starts a transaction on information of interest.

The following actions are available in this activity:

(a) Specify or edit a query and its execution context

- Query specification is discussed in activity (2) above. The execution context, also described above, consists of the domain containing the information of interest, the desired accuracy of execution results (i.e. recall and/or precision), and whatever indexes to be used for execution efficiency.

(b) Specify or edit the event schedule for executing an agent

- This involves selecting the events (e.g. Wednesday) and the schedule (e.g. every Wednesday) that will cause an agent's execution and registering the agent with a scheduler that will monitor Information Habitat events and invoke the appropriate agents.

(c) Describe and edit the information processing desired [may require an expert]

- As mentioned above, this is essentially a programming task. The user interface for this action must provide some means of entering such a program. Minimally, this will consist of a text editor (syntax-directed?) for a "little language". *Maximally, there might have to be a trapdoor here to a general-purpose programming language.*[10]

[10] Research into visual programming may be applicable here.

(d) Save agent description

- It will be possible to build up libraries of useful agents. Note that a library is simply some form of collection structure on a set of objects, in this case, agent specifications. We expect that users will "trade" useful agents with each other.

(e) Browse and search agent libraries

- As in the other cases, this involves applying activity (1) above (interacting with structured information) to libraries of agent specifications. Once an agent specification is selected, it can be edited using action (a) or managed using action (f).

(f) Manage agents

- This simply involves activating or deactivating a specific agent (i.e. in effect, establishing or removing an authorization for that agent to execute).

(8) Specify and Execute Transformations [expert user entry point]

Transformations are necessary whenever information structured one way needs to be viewed as structured another way. The need for transformations arises in several ways:

(1) Analysis -- Information in human readable form (i.e., text) is converted to an internal form. For instance, a text may be converted to a concordance or an index constructed of certain key terms. Analysis is essential not only for extracting structure from text but also for managing information kept in foreign data formats.

(2) Rendering -- Information in internal form is converted to text or back to a foreign data format.

(3) Transforming -- Not only will end-users wish to change their view of structured information (e.g. viewing a book's table-of-contents as an alphabetized index), but application builders will also need to specify how to transform the results of a query into the form required by the set of InfoActors to be used for interacting with those results (refer back to action (c) of the section on creating new applications above).

The last point is illustrated in Figure 2. That is, before any application for viewing and editing information (activity (1)) can be executed, the InfoActors comprising that application must be instantiated. Transformations specify how the application controller is to extract that data from given information (such as that returned by a query). Similarly, transformations indicate how to reconstitute information from the data held by an InfoActor so that modifications can be stored back in a database. If that database is a foreign data format, then analysis and rendering transformations will be required.

Certainly, end-users can also use a transformation facility to automate classification and other personalized structuring (e.g. previewing mail messages and making best guesses as to their content by classifying them into different "in" folders). Thus, transformations will be useful in the specification of agents (see activity (7) above).

The following actions are available in this activity:

(a) Specify and edit transformations

- To go from text (or foreign data format) to structure, some form of parsing must be specified. The specification consists of a grammar with embedded instructions for entering the parsed components of text as parts of the internal form. The technology used for parsing is described in more detail in the subsequent report on the system architecture [Horowitz 92].
- To specify standard structure-to-structure transformations, the user will not only have to indicate what happens to the source structure but will also have to select appropriate target structure, much as in activity (3) above (i.e. selecting and imposing structure). Again, the technology used for these kind of transformations is described in more detail below.
- Rendering transformations may be generated with appropriate string construction facilities of an extension language (*another trapdoor*).

(b) Save transformation [knowledgeable user]

- It will be possible to build up libraries of useful transformations. Note that a library is simply some form of collection structure on a set of objects, in this case, transformations.

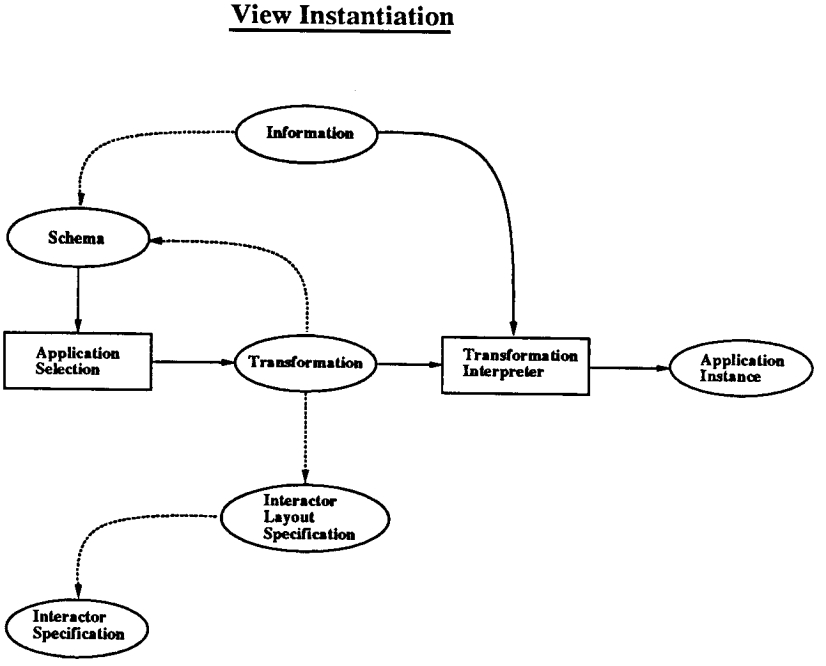

Figure 2: The Instantiation of an Application based on the type of Information

(c) Browse and search transformation libraries [knowledgeable users]

- This action applies activity (1) above (view, edit, and interact with information) to libraries of transformations. Once a transformation is selected, it can be edited using the capabilities presented by action (a) above or applied using action (d) below.

(d) Execute transformation

- In general, transformations will be executed implicitly by the system during application instantiation. Changing the way information is structured from one organization into another, however, will require an explicit action on the part of the user.

  To execute a transformation, the user must specify the information to be transformed and what to do with the resulting information. Specifying the information to be transformed may involve existing information or some computation that can act as an information source. In the latter case, the transformation acts as a filter.

(9) Create and Impose Indexes [knowledgeable user entry point]

In this activity, users impose a kind of structure on information purely for performance reasons. That is, the structure itself is not meant for browsing. For now, we expect that only text indexes will be provided in the system, since we do not know how to index arbitrary multi-media. The user may choose from a set of indexing technologies known to the system; although we hope to make this set extensible, we do not expect anyone but wizards to be able to create a new indexing technology *(a trapdoor is required here)*.

The user interface for this activity might be integrated with several of the other activities, especially since the user must select the set of information entities to be indexed.

The following actions are available in this activity:

(a) Select the collection of information entities to be indexed
- The user must not only specify the structure representing a collection of information entities, but must also indicate which textual attributes should participate in the indexing process.

(b) Select and apply an indexing technology
- Each technology may have a set of parameters the user can specify that affect the performance of subsequent retrievals (i.e. speed, recall, precision). To extend the set of known technologies, then, the implementor of a new technology must provide the appropriate user interface to be presented when a user selects that technology.

(c) Updating an index
- It is important for an index to be up-to-date with the collection it indexes. However, recomputing indexes can be a very time-consuming operation, and the user should have control over when an index is recomputed. By default, when an index is created, a re-indexing agent is also created which has four modes:
  1) re-index only when the user asks for a fresh index;
  2) when attempting to use a "stale" index, ask if the index should be recomputed;
  3) re-index periodically (e.g. nightly); and
  4) re-index whenever the collection changes.

Conclusions

As more and more information becomes available on-line, traditional mechanisms provided on computers for access and management of that information will start to break down. We believe that the solution to this problem lies not only in improved access technologies (i.e. better indexing and query algorithms), but also in providing new, highly extensible capabilities that allow users to organize the information of interest to them. These capabilities hopefully model the activities users employ[11] outside the computer to manage their information space today. In particular, the goals of Alexandria include enabling on-line maintenance of the personalized organization users impose on their information and enhancing those organization activities in ways that can only be achieved on computers (e.g. facile indexing and search).

We have identified several activities that would be useful for personally organizing large amounts of information. First, the ability to index and query information domains is clearly necessary. Personalized organization, however, implies that the user must be able to impose additional, structural information on top of existing information. Thus, at the minimum, a reasonable set of useful structures should be made available for such imposition (e.g. hierarchies, lists, cross-references). Once structure has been imposed, the ability to browse that structure and its corresponding information together transparently becomes essential. The desire for collaboration implies that users should be able to share such views, as long as adequate protection is provided. Thus, an adequate authorization facility should be built.

The need to deal with all kinds of information implies that parsing, rendering, and other transformation capabilities be integrated within an environment. Transformations also allow the same information to be viewed in different ways. Since transforming and imposing structure on large amounts of information by hand would be impractical, some means for creating background agents to do the work is important. And, of course, once background processing is made available, it can be used for other things as well, such as monitoring a dynamically changing information source and informing the user when changes of interest occur.

Finally, since the set of structure types and the applications that operate on instances of those types cannot be exhaustive within the environment, it must be possible to integrate in third-party applications, create new structure types, and create new applications that manage the information of those new types. The integration of a third-party application may be improved if it supplies a command language that can invoke the capabilities present within the Information Habitat. The construction of new applications can take advantage of the observation that most user-level actions of such applications are common to all information management applications. Thus, it should be possible to allow users to construct their applications interactively.

In fact, most of these activities admit to reasonable direct-manipulation user interfaces. Thus, we believe that the capabilities of the Information Habitat should be highly accessible to novice users. We intend to prove these assertions by building the Information Habitat and applying it to a specific user community that deals with information intensively and has access to large amounts of on-line information.

This paper described the motivations behind the research and the approach we propose to take. One or more companion papers will address the issues concerning the designs of the different capabilities to be provided in the Information Habitat.

[11] Or wish they could employ.

References

[Alashqur 89] A. M. Alashqur, S. Y. W. Su, H. Lam.
OQL: A Query Language for Manipulating Object-oriented Databases.
In *Proc. 15th International Conference on Very Large Data Bases*, pp. 1-10, August 1989.

[Bentley 86] J. Bentley.
Programming Pearls: Little Languages.
Communications of the ACM, 29(8), 711-721, August 1986.

[Cardelli 87] L. Cardelli.
Building User Interfaces by Direct Manipulation.
Technical Report 22, Systems Research Center, Digital Equipment Corp., October 1987.

[Chen 76] P. P. Chen.
The Entity-Relationship Model -- Toward a Unified View of Data.
ACM Transactions on Database Systems 1(1):9-36, March 1976.

[Carlson 89] C. R. Carlson, W. Ji, A. K. Arora.
The Nested Entity-Relationship Model - A Pragmatic Approach to E-R Comprehension and Design Layout.
In *Entity-Relationship Approach to Database Design and Querying, Proceedings of the Eighth International Conference on Entity-Relationship Approach, October 1989*, ed. F. H. Lochovsky, North-Holland, 1990.

[Fishman 87] D. H. Fishman, et. al.
Iris: An Object-Oriented Database Management System.
ACM Transactions on Office Information Systems 5(1):48-58, January 1987.

[Horowitz 91] M. L. Horowitz.
An Introduction to Object-Oriented Databases and Database Systems.
ITC Technical Report, ITC-TR-91-103, August 1991.

[Horowitz 92] M. L. Horowitz, et. al.
The Alexandria Project: Design, Architecture, and Implementation.
To be published as an ITC Technical Report, 1992.

[Palay 90] A. Palay, D. Anderson, M. Horowitz, M. McInerny.
The Alexandria Project: In Search of a Unified Environment for Information Access and Management.
ITC internal report.

[Rabitti 88] F. Rabitti, D. Woelk, W. Kim.
A Model of Authorization for Object-Oriented and Semantic Databases.
In *Proceedings of the International Conference on Extending Database Technology*, pp. 231-250, March 1988.

[Rubine 91] D. Rubine, G. Keim.
SYSTEM: A Multimedia Information System.
ITC internal report.

[Shan 90] Y-P. Shan.
MoDE: A UIMS for Smalltalk.
OOPSLA/ECOOP '90, SIGPLAN Notices 25(10), 258-268, October 1990.

[Su 89] S. Y. W. Su, V. Krishnamurthy, H. Lam.
An Object-oriented Semantic Association Model (OSAM*).
In *Artificial Intelligence: Manufacturing Theory and Practice*, ed. S. T. Kumara, A. L. Soyster, and R. L. Kashyap, Chapter 17, Institute of Industrial Engineers, Industrial Engineering and Management Press, Norcross, GA, 1989.

[Teorey 86] T. Teorey, D. Yang, J. P. Fry.
A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model.
ACM Computing Surveys, 18(2), 197-222, June 1986.