Designing structure-based information agents

Daniela Rus and Devika Subramanian
Department of Computer Science
Cornell University
Ithaca, NY 14853

Abstract

This paper presents a new methodology for building special-purpose software agents that capture and access information in large, heterogeneous, distributed information environments. It allows rapid prototyping of information agents for solving a wide range of retrieval tasks with guarantees on performance. The key idea is to exploit underlying *structure* at various levels of granularity to build partial models that act as high-level indices with task-specific interpretations. These partial models are constructed using modules called navigators. Information agents are configured by using effective communication protocols to connect structure detectors and navigators. This methodology is applied to the design and implementation of information agents in two contexts: one for retrieving stock market information from scanned copies of newspapers and another for retrieving technical reports from the Internet.

1 Introduction

The proliferation of information in electronic form and the development of high-speed networking makes the problem of locating and retrieving information in vast electronic environments one of the grand challenges of artificial intelligence. Our research goal is to develop methods for solving the *information capture and access* (ICA) problem in heterogeneous, distributed environments. In its most general form, the ICA problem can be posed as follows: given an electronic data environment, capture it by acquiring partial models of it and access it guided by user-specified tasks. The modeling of the data environment associates task-level content with the information which in turn facilitates location and retrieval. Examples of data environments are electronic repositories of newspapers and technical reports, data from high-energy physics experiments, weather satellite data, and audio and video recordings. Examples of tasks that query and manipulate environments include content-based retrieval of technical reports, access of documents via citations, summaries of stock prices from archives, and retrieval of temporal weather patterns from a weather database.

A diverse collection of tools like WAIS [Kah91], Gopher, Archie and Mosaic have been developed to provide keyword-based access to large ASCII text environments, as well as limited manipulation (e.g., display) of non-textual information. Underlying these tools are two assumptions about the nature of ICA tasks and methods for decomposing them. The first assumption is that tasks can be specified with words that occur in the data environment. The second assumption is that words form a natural index into the environment. Tasks for which word-based indices are inadequate or do not exist, are either not supported, or are solved by scanning every word in the data environment. Consider the retrieval request "get the paper by Tom Mitchell with the picture of the Theo architecture on the second page". This request cannot be handled reliably by simple keyword-based systems. Yet another example is the query "find a mechanism that converts a uniform rotary motion into a reciprocation in Artobelovsky's design encyclopedia [Art79]". To answer this query, recognition of kinematic function is necessary.

In this paper, we propose a methodology that relies on detectors of structure to identify entities at an appropriate level to use as indices into the data environment. These high level structural units function as beacons or landmarks in the data environment. We call this approach *structure-based information capture and access*[1]. Structure-based ICA decomposes the retrieval problem into recognition of high-level structure and traversal of portions of the data environment with this structure. By structure, we mean any level of abstraction higher than the basic unit of data representation like characters and pixels. For example, tables, figures, lists, paragraphs and sections are layout-based abstractions for documents. Theorems, lemmas, examples and counterexamples are content-based abstractions. These structures encode semantic information about the data, and serve as filters to select the portions of the data relevant to a task. These structures, however are not immediately available. We develop *information agents* to actively detect task-specific structure in the environment and exploit them for efficient retrieval.

Consider the task of finding precision recall measures for specific collections (such as the CACM) from a scanned set of technical reports. Precision recall numbers are contained in articles on information retrieval and they are frequently displayed in tables. Figure 1 shows a zoomed-out view of a paper on informa-

[1] Conventional word-based systems are also structure-based ICA systems. However, they exploit only one type of structure in the data environment.

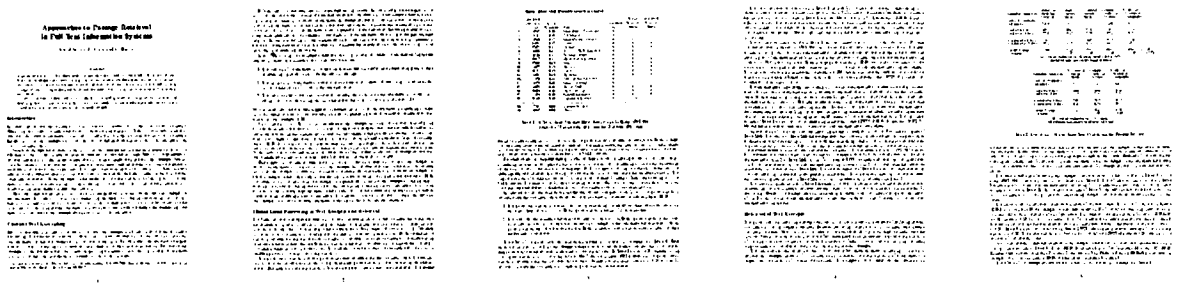

Figure 1: A zoomed-out view of an article on Information Retrieval

tion retrieval that highlights the presence of tables. In our approach, we view the pages of the article at a coarse granularity to pick out tables, using simple geometric computations. We then examine the tables at a finer level to extract the desired information. Our approach is effective in information domains with data is represented at a level of detail that is too fine for exhaustive searches to be feasible (*e.g.*, bits, for scanned text), but where there is enough hidden, high-level structure to filter relevant information efficiently.

We implement our generic task decomposition scheme using information agents [KC88, 8]. An information agent is a composition of special-purpose programs called *structure detectors* and *navigators*. Structure detectors efficiently decide whether a block of data has a specified abstract property to within specified error tolerance and confidence, *e.g.*, "is this block of text a table or a graph?" Navigators decompose the data segments extracted by structure detectors into more detailed units. For instance, the table navigator for the precision-recall agent extracts models of the table as a set of rows and refines it as a set of characters in each row.

Underlying this task decomposition is the belief that (1) high-level structures such as tables and trajectories can be robustly detected by simple and efficient programs with guaranteed performance properties, (2) a library of parametric structure detectors for a variety of tasks and environments can be constructed, and (3) navigators that extract portions of the data environment with the detected structure can be constructed. All of these beliefs can be put to empirical test. In this paper we build information agents based on this task decomposition for two retrieval tasks with very different characteristics.

Information agents roam their environment, aided by their navigators, looking for patterns, recognized by their structure detectors. This approach to the ICA problem bears a strong similarity to the search of physical environments by mobile robots [4]. Structure detectors are virtual sensors and navigators are virtual effectors. The detailed structure of an information agent is shown in Figure 2. It is a tree of alternating layers of structure detectors and navigators

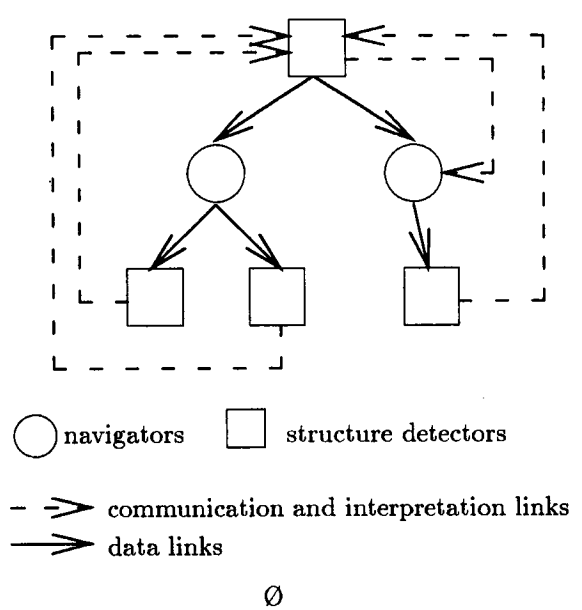

Figure 2: The architecture for Information Capture and Access. The tree alternates structure detectors and navigators.

glued together by communication paths. The root of the tree is a structure detector for the user's query. This alternating structure implements the zoom-in, zoom-out strategy used in our example in Figure 1. Finer and finer partitions of the data environment are examined by the levels further away from the root. There are two types of paths between nodes. The first type flows downwards through the tree, conveying data from structure detectors to navigators at successive levels. In addition to these data paths, an information agent also has communication and interpretation links, that can connect any two nodes in the tree. The communication and interpretation paths are used for answer assembly, and for error detection and recovery. We construct these agents from a library of robust structure detectors and navigators that we have built [RS94]. Our ultimate goal is to assemble these agents automatically from task specifications using classical planning techniques.

2 Modular Agent Architectures

In this section, we describe navigators and structure detectors in more detail, and provide examples of each. We also discuss methods of composing information agents from these components with appropriate data and control flow paths.

2.1 Navigators

Navigators are programs that actively that make "maps" at the various granularities of representation. Each map is a partial model of the world. The representation of the article in Figure 1 at the paragraph level is such a model. Navigators segment the data in the environment at the appropriate grain size for structure detectors. We model granularity shifts in the descriptions of the data environment using concepts from topology[2] [Mun75].

Definition 2.1 *A navigator n is a function that takes a topology of the world W and produces a refinement of that topology.*

$$n : \tau_1(W) \rightarrow \tau_2(W) \quad \textit{where } \tau_2 \textit{ is a refinement of } \tau_1$$

This abstract specification of a navigator allows us to characterize its behavior independent of its implementation. The set $\mathcal{T}$ of all topologies of a data environment W is computed from a set $\mathcal{N}$ of navigators as a finite (functional) composition of elements of $\mathcal{N}$. Since the refinement relation between topologies is a partial order, $\mathcal{T}$ is a lattice. The top element of the lattice $\mathcal{T}$ is the trivial topology of W. The bottom element $\perp$ is a topology consisting of singleton subsets containing the elements of W. Modeling the data environment as a lattice of topologies generated by navigators provides computational support for the idea of generating successive refinements that zoom-in to the required information.

We now present a logical navigator that segments a scanned document, and characterize its efficiency and robustness. Given a pixel array of a document, the goal is to partition the document into regions that capture its layout. In what follows, we present an algorithm for finding specific regions of layout structure.

Definition 2.2 *Let B be a polygonal segment of the document. A border of width d is the set $Border(B, d) = B \oplus S^1_d - B$. Every element of $Border(B, d)$ is a white space*[3].

The geometric definition of a border is parameterized on d, the width of the border. It allows us to construct a hierarchical model of the document layout without relying on a predefined set of document features like sections, captions, *etc.* The levels of the hierarchy are defined by different values of d. That is, we can partition document into regions at different grain sizes according to the border of white space that can be drawn around them. For example, if the task is to partition the front page of The New York Times into blocks, values of $d > 0.2$ inches extract the entire title, while values of $d < 0.1$ inch separate each character of the title into its own block. Thus a collection of d values defines a totally ordered set of region topologies for a document. Each topology is computed by rolling coins of successively smaller sizes through the maze of white spaces in a document [RS94].

To make the border definition apply to real documents where runs of white pixels are polluted by black pixels that occur at random locations, we associate a tolerance parameter ϵ_d, with every d value. ϵ_d represents the number of black pixels that can be ignored in the detection of a run of white pixels of length d. Metaphorically speaking, we treat each coin of size d as a bulldozer that can push ϵ_d or fewer black pixels out of the way. In our implementation of the block segmenter, we experimentally determined these values for documents drawn from the scanned technical report archive at Cornell.

Our block segmentation algorithm finds regions by traces their perimeters. Existing algorithms for block segmentation, for example the *run length smoothing* [WCW82] and the *recursive x-y cuts* procedure [NSV92], examine every pixel in the document. Our perimeter tracer, in contrast, only examines pixels that are outside each region. Its complexity is $O(p)$, where p is the number of pixels that do not occur in any of the identified regions. For each region, the complexity of our algorithm is linear in the perimeter of the region, rather than its area. For dense documents, like lead pages of a newspaper, this is a significant reduction in complexity.

The correctness of our block segmentation scheme relies on regularities in the environment of documents – in particular, the fact that documents are typeset with some standard conventions. Most documents have a rectangular layout produced with a finite set of fonts. Each font size has a characteristic spacing between lines and characters. Suppose the inter paragraph spacing is d_p, and we supply the algorithm with a d value of $d_p \pm \epsilon_p$, where $0 \leq \epsilon_p \leq d_p$. How likely

[2] Let τ_1 and τ_2 be two topologies over a set S. If $\tau_1 \subset \tau_2$ then τ_2 is refinement of τ_1.

[3] $A \oplus B = \{a + b \mid a \in A, b \in B\}$ is the Minkowski sum of sets A and B. S^1_d is a circle of diameter d.

are we to extract paragraphs in the scanned document using the block segmenter? The answer to this question is determined entirely by the data environment and not the algorithm. Our algorithm is robust in environments where significant d values are spaced more than ϵ_p apart. A more formal statement of this intuitive analysis of robustness requires the formulation of layout models of scanned documents. A formal robustness analysis is presented in the next subsection for a structure detector that operates on paragraph-level regions in an ASCII document.

2.2 Structure detectors

Structure detectors are programs that can efficiently decide whether a block of data has a specified property P. An example is the property of being a table or a graph for a block of text.

Definition 2.3 *A structure detector is a computable function s defined on a topology $\tau(W)$ of the world to a* discrete *subset $t(W)$ of that topology, such that $t(W)$ has the property P.*

$$s : \tau(W) \rightarrow 2^{\tau(W)}$$

$\tau(W)$ is a topological set. Text, numeric and digital audio and video data environments are topological sets, due to their discrete nature. A structure detector s for a property P is *correct* if it finds all the subsets of $\tau(W)$ which satisfy P. A structure detector s for property P is *robust* if whenever it recognizes $t(W)$, it recognizes all its ϵ-perturbations. We now describe the design of a robust table detector.

Webster's Seventh Dictionary defines a table as a *systematic arrangement of data usually in rows and columns for ready reference.* Implicit in this definition is a *layout* component and a *lexical* component: the data is organized in columns of similar information. Consider the table in Figure 3: while its layout and lexical structures are clear; it is not very regular. Our goal is to create a table detector that checks for column and row structure while tolerating irregularities to within specified error bounds.

The measure for the column layout of a block of text is given in terms of a data structure called the *white space density graph* and denoted by WDG. Let B be a block of text of n rows and m columns and $w : \{c|c \text{ is a character }\} \rightarrow \{0,1\}$ with $w(space) = 1$ and $\forall c \neq space, w(c) = 0$.

Definition 2.4 Vertical structure: *The* white space density graph *of B is the polygonal line* WDG : $[0,m] \rightarrow [0,1]$ *defined by the points* $\text{WDG}(i) = \frac{1}{n}\sum_{j=0}^{n} w(B_{i,j}),\ 0 \leq i \leq m$

Definition 2.5 Deviations in vertical structure: *Given an error tolerance ϵ_v, a block of text has* column structure *if it occurs between two successive local maxima in the* WDG *above* $(100 - \epsilon_v)\%$.

Each local maximum in the WDG is a candidate column separator. A candidate column is a real table column only when it has corresponding horizontal lexical structure. We are far from being able to identify row structure based on semantic content, but semantic uniformity in rows is highly correlated with lexical uniformity. We exploit this correlation in the design of a robust table detector. In distinguishing lexical structure, we identify the following equivalence classes of characters: alphabetic, numeric, and special (each special character is in its own class). Let $c_0, c_1, \ldots c_n$ denote the columns of a table. We use regular expressions for generalizing the contents of a column.

Definition 2.6 Horizontal structure: *Consider the columns $c_1 \ldots c_n$ of a block of text, and consider the lexical descriptions $r_1 \ldots r_n$ of these columns. This text also has* row structure *if and only if the language described by $r_1 r_2 \ldots r_n$ is non-empty.*

Given $\epsilon > 0$, two strings a and b are ϵ-similar if $M(a,b) \leq \epsilon_h$, where M is the Levenshtein metric for string comparison.

Definition 2.7 Deviations in horizontal structure: *Given $\epsilon_h > 0$ and a set of strings, an ϵ_h-*typing *is a division of the set into disjoint subsets such that any two strings in the same subset are ϵ_h-similar.*

Lexical typing for a table is done in two parts. Each candidate column is analyzed to determine a regular expression for its type. The alphabet of types is generated by ϵ_h-typing the column elements. The lexical type of the table is obtained by computing the minimum regular expression over the column types. This step allows for the occurrence of multi-line records in a table and for ϵ_h tolerance in the record units. A minimal ϵ_h-typing partitions the elements of the column in the coarsest possible way.

We analyze the robustness and performance of the algorithm in [RS94].

2.3 Gluing detectors and navigators by communication

Information agents are created from task specifications. In this section, we discuss how simple agents are constructed from available detectors and navigators, and how complex agents can be built from simple ones. Synthesizing an information agent for a given task consists of (1) identifying a set of structures at different levels of detail that are relevant to the task, and choosing (or creating) detectors that can recognize these structures, (2) choosing navigators that segment data at the right granularity for each detector, (3) composing the agent tree from these navigator/detector pairs, and (4) interpreting the computed data.

A simple agent is composed of a navigator and detector connected in series. For this composition to make sense, we need the output topology of the navigator to match the input topology of the detector. We call the topology matching constraints *calibration* constraints.

Definition 2.8 *Simple agent: A simple agent is constructed from a navigator $n : \tau_1(W) \rightarrow tau_2(W)$ and a detector $s : \tau(W) \rightarrow 2^{\tau(W)}$ connected in series, denoted $n \bullet s$, such that the calibration constraint $\tau_2(W) = \tau(W)$ holds.*

```
100a Introduction to Computer Programming 4    Lec1 TR    9:05 Ives 120
     Wagner                                Lec2 TR   11:15 Ives 120
100b Introduction to Computer Programming 4    Lec1 TR    9:05 Olin 255
     Van Loan                              Lec2 TR   11:15 Ives 110
101  The Computer Age                      3        TR    1:25 Upson B17
     Segre
102  Intro Microcomputer Applications      3    Lec1 TR   10:10 RR 105
     Hillman (Ag. Engr.)                   Lec2 TR   12:20 RR 105
108  A Taste of UNIX and C                 1(4wk)    MWF   3:35 Olin 255
     Glade                          su
```

Figure 3: A Schedule of the Introductory Computer Science Courses

We define two composition schemes for agents and identify calibration constraints on each scheme.

Definition 2.9 *Serial composition: An agent* $a_1 : \tau_{in1}(W) \rightarrow \tau_{out1}(W)$ *can be serially composed with an agent* $a_2 : \tau_{in2}(W) \rightarrow \tau_{out2}(W)$ *in that order yielding a new agent* $a : \tau_{in1}(W) \rightarrow \tau_{out2}(W)$ *constructed from the functional composition of* a_1 *and* a_2*, provided the calibration constraint* $\tau_{out1}(W) = \tau_{in2}(W)$ *holds.*

Definition 2.10 *Parallel composition: An agent* $a_1 : \tau_{in1}(W) \rightarrow \tau_{out1}(W)$ *can be composed in parallel with an agent* $a_2 : \tau_{in2}(W) \rightarrow \tau_{out2}(W)$*, yielding an agent* $a : \tau_{in1}(W) \rightarrow \tau_{out1}(W) \times \tau_{out2}(W)$*, provided the calibration constraint* $\tau_{in1}(W) = \tau_{in2}(W)$ *holds.*

If a_1 and a_2 are simple agents, then the above operation constitutes sharing of a navigator. Parallel composition of two simple agents allows for different recognizers to operate on the same data partition. For instance, a table detector and a detector for recognizing graphs in text both employ paragraph level partitions.

With these two composition schemes we can built agent trees in a task-directed manner. So far we have only discussed constraints on data flow between agents. We now turn to control flow. There are two approaches to control flow in the agent tree: the centralized control scheme, and the localized control scheme. Each of these is well-suited for different situations: Centralized control gives the root node control over passing partition parameters to the appropriate navigators. The root itself gets the parameters from the user. Localized schemes distribute invocation control throughout the tree.

The designer ensures that the agent maintains an invariant – the mapping between the results of computations (recall these are topologies) and the "meaning" of the extracted data. In the Stock Agent, discussed in the next section, the designer establishes a mapping between the structure extracted by the table detector and a "stock table" – the latter is a task-specific category, the former a geometric object. The mapping constitutes the interpretation of items in the rows as companies and stock prices, and items in the columns as high, low and closing values. The designer incorporates checks in each agent to ensure the integrity of this mapping between results of computations performed by the agent and their task-specific interpretations. Further examples of data interpretation are discussed in the context of the Stock Agent and the Bib Agents described in the following sections.

3 Example 1: Compiling Reports from Tabular Data

Consider the task of compiling a stock report for AT & T for a given period of time using an electronically archived collection of The New York Times. For each day in the given time interval, we can structure the task as follows:

1. We partition the paper into sections using the navigator in Section 2.1 with the border parameter that characterizes sections in The New York Times.

2. We filter the business section from the paper (this is where the stock data is most likely to occur) by using a structure detector for section titles.

3. We partition the business section into blocks at the paragraph level, using the navigator in Section 2.1 with the border parameter that characterizes paragraphs.

4. Since stock data in The New York Times is represented in tabular form, or in graphical form, we select the tables and graphs using the table detector and the graph detector.

5. We zoom into each table and graph to extract the specific information on AT&T.

The agent that solves the problem in this manner is shown in Figure 4. The reader can parse this as serial and parallel compositions of several simple agents assembled from detector/navigator pairs. The root of the tree is a structure detector that assembles the stock report on AT & T for the requested interval of time. The first level navigator is a block segmenter that takes a bitmap copy (one at a time) and produces its paragraph level partition. The structure detector that follows it eliminates all the blocks that are half-tone pictures using methods discussed in [WS89]. The remaining blocks are partitioned at the character level using an OCR structure detector. To maintain the alternating structure of the tree, the ASCII paragraphs generated by the character recognizer are passed through the identity navigator to our table detector. The subset of paragraph blocks that

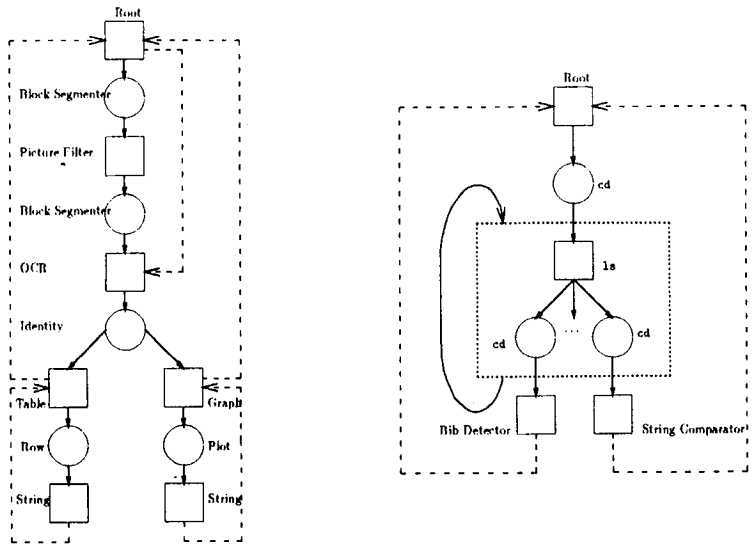

Figure 4: An information agent for compiling stock reports (left). The Bib agent (right).

are recognized as tables are provided to a row navigator, in order to separate the records. The rows are individually passed to a string comparator, to identify the AT & T records. The data interpretation phase is complex. The link between the string comparator and the table detector has knowledge of stock table layouts and is used for passing and interpreting the AT & T record. This knowledge, encoded as rules, is used to identify the location of specific columns, e.g., the closing column.

The link between the root node and the character recognizer is used to handle a common failure mode. If the closing value for a specific day is unavailable, the root invokes an alternate computation. For this problem, the blocks generated by the character recognizer are passed to a graph detector. This is because stock values are sometimes reported via graphs. The graph detector recognizes blocks containing graphs, which are decomposed into text and curves. Another detector which uses a string comparator identifies the curve corresponding to AT & T and extracts the desired value.

Note that even though the computation is local, communication paths vital for answer assembly and error recovery are non-local in this design. We are examining methods of localizing communication through the tree while still being able to provide powerful error detection and recovery strategies.

We have implemented the Stock Agent for the data domain of Internet newsgroups. Typical messages consist of a combination of prose and tables that vary in format. This agent extracts tables from a given list of business newsgroups which are then searched for records labeled "ATT". An extracted record is interpreted by rules that define the High, Low, and Closing columns on a table of stock data. Sample results for running this agent are given in Figure 5.

We note that the Stock Agent can be used to retrieve any other type of information that is present in tabular form, if augmented with appropriate data interpretation procedures. In particular, we have instantiated the design in Figure 4 for the task of retrieving precision-recall measures for specific collections from our database of scanned technical reports on information retrieval.

4 Example 2: Retrieving Technical Reports

With increasing amounts of heterogeneous distributed data, there is a need for tools that can autonomously obtain information from fairly coarse specifications, with minimal human intervention. In this section we describe an agent that retrieves technical reports and bibliographic references from the Internet given high-level specifications of the desired information. As an example, consider the query "retrieve recent articles on machine learning from the technical report ftp sites through the Internet". This query provides information about the possible locations of the articles, but more information is needed to find the actual subdirectories that contain articles on machine learning.

We have implemented Bib Agent to accomplish this task. Bib Agent makes guesses (based on incrementally gathered statistics) about the plausible location of the report or reference and autonomously navigates through the network looking for it. It also enhances its performance over time by remembering paths taken in previous searches. It deciphers the best possible way of presenting retrieved information to the user.

Bib Agent's structure is shown in Figure 4. Bib Agent is built on top of the Alex [Cat92] filesystem which provides users transparent access to files located in all the ftp-sites over the world. It uses Unix com-

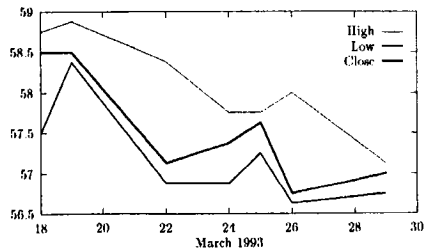

Figure 5: Compilation of a stock report for AT&T over time, using the stock agent in the environment of newsgroups

mands like cd and ls to work its way through the directories and subdirectories accessible by anonymous ftp. A structure detector that consists of the ls Unix utility and a filter selects a subset of directories for the next set of navigators. These directories are processed in parallel. Bib Agent uses a mixture of default path selection and human intervention to navigate its way to the desired information. We have a specialized structure detector with knowledge of bibliographic data files (.bbl and .bib files). This detector can efficiently retrieve a completed bibliographic reference using partial knowledge of some of the fields of a reference. The answers from the leaves of the tree are passed directly to the root, as they are computed. A sample output from our implementation is shown in Figure 6.

Bib Agent is a learning agent. It incrementally constructs a road map of the Internet indexed by queries. The road map consists of cached paths to information relevant to the query. Bib Agent also learns from user input as it searches. A user can thus customize Bib Agent with his own preferences by showing it some examples. The complexity of this agent arises from the considerable amount of procedurally encoded knowledge about the Unix file organization embedded in each structure detector in the tree.

5 Discussion

Our research goal is to develop and prototype a methodology for conceptual retrieval tasks in large, heterogeneous, distributed multimedia environments. In most data environments, content is partially encoded in extant underlying structures at varying levels of granularity. We exploit this regularity and propose the construction of information agents using structure detectors and navigators glued together by communication, as a paradigm for organizing the search for information.

We provide users with an assembly kit for constructing agents from a library of structure detectors and matching navigators, and communication glues. These detectors and navigators come with performance guarantees. For instance, the table detector can detect tabular information to within user-specified tolerance in row and column misalignments. Since the function of each component is clearly specified, users can assemble agents with the same ease as drawing schematics on paper.

Our design philosophy is a sharp contrast to that behind large, general-purpose systems like Wais, Archie, and Gopher. When is this design methodology more appropriate than others? To answer this question, we need a theoretical basis for characterizing ICA tasks and for measuring the effectiveness of alternate architectures on task classes. The notion of structure introduced in this paper is a first step toward characterizing tasks in an implementation-independent manner. We constructed an information agent for the class of retrieval tasks whose answers can be found in tabular form. We instantiated this agent for compiling stock reports and for finding precision-recall measures. We built another information agent using the same paradigm for a task that required physical navigation over the Internet. On the experimental side, we are at the beginning of a long road. We need to integrate our work with widely-used multimedia data. Our current work with the CS-TR project, a nation-wide effort to create an electronic library of computer science reports, will serve as a large-scale testbed for information agents.

References

[AS93] J. Allan and G. Salton, The identification of text relations using automatic hypertext linking, in *the Workshop on Intelligent Hypertext, the ACM Conference on Information Knowledge Management*, November 1993.

[Art79] I. Artobolevsky, *Mechanisms in Modern Engineering Design, vols. 1-4*, MIR Publishers, Moscow, 1979.

[BC92] N. Belkin and W. Croft, Information filtering and information retrieval: two sides of the same coin?, *Communications of the ACM*, vol. 35(12), pp. 29-38, 1992.

[Bro86] R. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, 1986.

[Cat92] V. Cate, Alex: a global file system, in *Proceedings of the Usenix Conference on File Systems*, 1992.

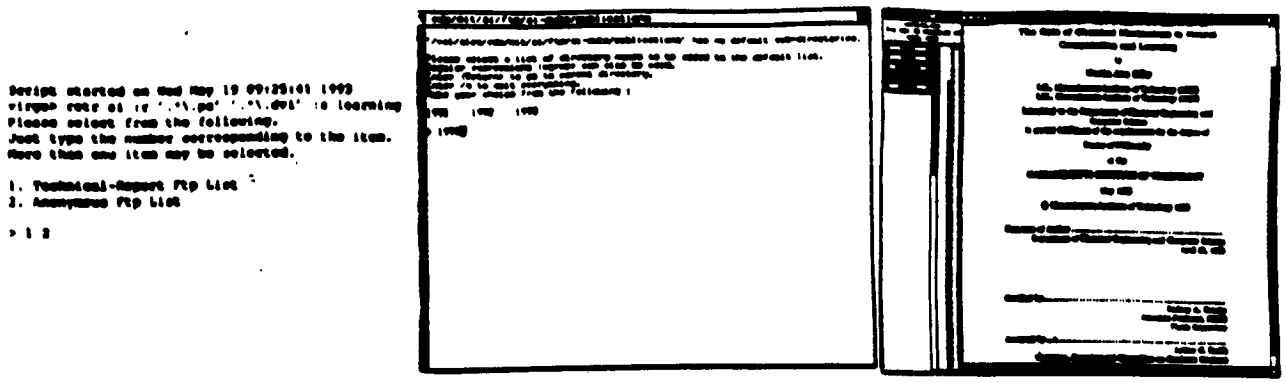

Figure 6: The user interface of Bib Agent. The left image is the invoking command. The agent is asked to locate papers on *learning*, by looking at *.ps* and *.dvi* files in repositories of technical reports on *ai*. The agent interacts with the user by giving two choices: *the Technical Report ftp list* and *the Anonymous ftp list*. The user selects both. The middle image represents the next interaction between the agent and the user, which happens when the agent searches the MIT repository. The agent automatically navigates to the *mit/ai/ftp/ai-pubs/publications* directory but does not know how to select between the *1991*, *1992*, and *1993* subdirectories. The user selects 1993, and the agent returns the only paper that includes the word *learning* on the first page. The first page of this document is shown on the right.

[Don94] B. Donald, Information Invariants in Robotics, to appear, *Artificial Intelligence*.

[DJR93] B. Donald, J. Jennings, and D. Rus, Information Invariants for Cooperating Autonomous Mobile Robots, in *Proceedings of the International Symposium on Robotics Research*, 1993.

[ES92] O. Etzioni and R. Segal. Softbots as testbeds for machine learning, in *AAAI Spring Symposium on Knowledge Assimilation*, 1992.

[GGM93] L. Gravano, H. Garcia-Molina, and A. Tomasic, The Efficacy of GlOSS for the Text Database Discovery Problem, Technical Report no. STAN-CS-TN-93-01, Computer Science Department, Stanford University, 1993.

[HP93] M. Hearst and C. Plaunt, Subtopic Structuring for Full-Length Document Access, in *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pg. 59-68, 1993.

[Kah91] B. Kahle, Overview of Wide Area Information Servers, WAIS on-line documentation, 1991.

[KC88] R. Kahn and V. Cerf, *The World of Knowbots*, report to the Corporation for National Research Initiative, Arlington, VA 1988.

[Mun75] J. Munkres, *Topology: A First Course*, Prentice Hall, 1975.

[NSV92] G. Nagy, S. Seth, and M. Vishwanathan, A prototype document image analysis system for technical journals, *Computer*, vol. 25, no. 7, 1992.

[RS93] D. Rus and D. Subramanian, Multi-media RISSC Informatics: Retrieving Information with Simple Structural Components, in *Proceedings of the ACM Conference on Information and Knowledge Management*, Nov. 1993.

[RS94] D. Rus and D. Subramanian, Information retrieval, information structure, and information agents, submitted to *ACM Transactions on Information Systems*.

[RSa93] D. Rus and K. Summers, Recognition and interpretation of electronic document layout, submitted to the 1994 IEEE Multimedia Conference.

[SEKN92] M. Schwartz, A. Emtage, B. Kahle, and B. Neuman, A comparison of Internet discovery approaches, *Computer Systems*, 5(4), 1992.

[WS89] D. Wang and S. Srihari, Classification of newspapaer image blocks using texture analysis, *Computer Vision, Graphics, and Image Processing*, vol 47, 1989.

[WCW82] K. Wong, R. Casey, and F. Wahl, Document Analysis System, *IBM Journal of Research and Development*, vol. 26, no. 6, 1982.