Multi-media RISSC Informatics:
Retrieving **I**nformation with **S**imple **S**tructural **C**omponents

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

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

Abstract

This paper presents a novel architecture for building special-purpose agents for solving the information capture and access problem in large, unstructured data environments with uncertainty in data representation and interpretation. The key idea is to recognize and use underlying structure *in such environments using* simple components *with specified performance guarantees connected by effective communication protocols. This paper brings together ideas from systems and theory: the design of mobile robots in unstructured physical environments, work in electronic libraries and knowbots, topology, as well as the rich research in information retrieval and filtering to design a concrete software architecture for solving a wide range of high-level user queries in multi-media environments.*

1 Introduction

The *Information Capture and Access* problem, in its most general form is posed as follows: given an electronic data environment, capture it by acquiring partial models of it and access it guided by tasks posed by users. We are interested in handling large amounts of unstructured data distributed over a wide area network. The tasks are querying and manipulating the data environment.

Examples of environments we consider include electronic libraries, satellite weather data, data from high energy physics experiments, *etc.* Even though these domains have data with inherent uncertainty in representation and interpretation, they have enough underlying structure to support the design of efficient capture and access algorithms. Our notion of structure is any level of abstraction higher than the basic unit of data representation. We need high level abstractions because the base-level information is too detailed for even linear-time algorithms to be feasible. Characters, pixels, and numbers are the basic units of raw data. We conceive a hierarchy of structures with task-level semantic content that is grounded in the basic data representations, for example, tables and figures, color coded regions, and trajectories of particles. These structures encode high-level information about the data but they are not immediately available. However, they can be actively acquired according to the type of task and used to construct partial models of the environment. The partial models filter the portions of the data relevant to the task. To construct effective partial models, we need to relate the structure of the environment to the structure of the task.

We believe that high-level structures such as tables and trajectories can be robustly detected and extracted by simple and efficient special-purpose programs with guaranteed performance properties [18]. These programs, called *structure detectors*, generate a *zoomed-out* view of the data by highlighting the presence of specific coarse-grained structures. For instance, an electronic page can be segmented into paragraphs (blocks) by bands of white space wider than than the inter-sentence spacing. This is an "out of focus" view of the page, since we pay no attention to any aspect of the interiors of the paragraphs. Given a block that potentially contains information of interest, a *zoomed-in* view produced by a finer-grained analysis extracts the required data. Consider the task of finding the current *AT&T* stock price using an electronically archived copy of today's The New York Times. We structure the task as follows:

1. We construct a zoomed-out view that extracts the blocks in the document.

2. Since stock data in The New York Times is represented in tabular form, we identify the blocks with tables in them.

3. We zoom into each table to extract the specific information on *AT&T*.

The first and third tasks are accomplished by special-purpose programs that we call *navigators.* A table detector performs the second task.

A variation of the stock report task is to compile a report on the evolution of the *AT&T* stock over the past year. There are two approaches for doing this. One is to sequentially repeat the algorithm outlined in the previous paragraph for every day of the year. Another approach is to scan these issues in parallel.

Consider a more complex variation of the stock market problem, where all the relevant information sources are not specified. Solving this problem requires navigating through the provided electronic documents, and extracting citations to other potentially relevant sources. Each source is then processed by a sequence of zoom-out and zoom-in operations. Planning for efficiently acquiring the relevant information sources can be done automatically, and is an important component of the Information Capture and Access problem.

Structure detectors and navigators naturally decompose the problem of information capture and access into recognition and traversal. This decomposition gives us the computational support for the implementation of the zoom-out, zoom-in strategy that takes advantage of the existing structure in the data at various levels of granularity. In the stock market example, structure detectors for *section headings* extract the financial pages for navigators that segment pages into blocks. Table detectors filter blocks that contain tables and pass them to navigators that segment the data for *word* level extraction. Navigators participate in the zoom-in process by filtering information for detectors at the right level of detail. Structure detectors and navigators are composed to form RISSC agents that compute answers to user queries.

RISSC agents has been derived on the basis of our experience in using wide-area information servers like WAIS [15] and Gopher. These systems allow a user access to very large ASCII text documents. Queries are posed by specifying keywords[1]. When word-based indices do not exist, queries are either not supported, or are answered by matching a regular expression over characters by scanning every character in the collection of documents. This low-level access to data limits the class of tractable queries. With increasingly rich sources of information available over the network in the form of scanned documents, audio, and video, there is a need to enlarge the scope of queries and access pieces of documents by structural units larger than individual words. Take the query "fetch the table that summarizes the closure properties of families of languages from *Introduction to Automata Theory, Language, and Computation* by John Hopcroft and Jeffrey Ullman." A pure text-based system augmented with good indexing fails to retrieve the table. As another example, consider the retrieval request "get the paper by Bruce Donald with a photograph of Tommy in it". These requests can be handled by structure detectors that can identify higher-level entities like tables and photographs. Yet another example is the query "find a mechanism that converts a uniform rotary motion into a reciprocation in Artobelovsky's design encyclopedia [1]". Paper encyclopedias are typically indexed by the component types in the mechanism, and not by behavior. To answer this query, a structure detector that embodies knowledge of kinematics is used, much like the structure detectors for the previous examples that embody knowledge of document layout.

Structure detectors are like band pass filters which extract information within specific frequency ranges. A library of structure detectors pulls out different bands of interest. The price to pay is that there will be bands that will not be recognized, but the benefit is the filtering. Structure is a high-level cue for content. We are proposing to exploit the existing but hidden structure in a large data environment to automate, speed-up, and enrich the access process. We are not aiming to answer every possible query by automatic generation of order from chaos. The scope of retrieval addressed by our architecture is restricted to classes of problems and environments for which structure detectors and navigators can be constructed. Examples of such environments are libraries of scanned documents, newsgroups, electronic mail, weather maps, *etc.* Retrieval methods like keyword indexing do not take advantage of the presence of a multiplicity of structure, and thus provide limited access into such worlds. Our methodology can thus be viewed as the on-line creation of multi-level indices, as in automated document structuring [27, 9, 19].

1.1 Previous Work

The proliferation of information in electronic form, the development of high-speed networking, and the evolution of digitized video and audio, makes the retrieval of relevant information from imprecise query specifications one of the grand challenges of computer science. Tools like WAIS [15], Gopher, Netfind [26], Archie, Mind [7], and CORE [17] have been developed in response to this challenge. One of the more detailed proposals for organizing architectures for retrieving information from electronic repositories is [16]. This work provides the context for the problem addressed in this paper. We are inspired by research in four distinct areas: information retrieval and filtering, mobile robotics, knowbots as proposed in [16], and automated document structuring.

- *Mobile Robotics.* The analogy between mobile robots in unstructured physical environments and information agents in a rich multi-media data environments is not just metaphorical. We have observed that the lessons [3, 4] learned in designing task-directed mobile robots can be imported to the problem of information capture and access. Section 2.2 provides a detailed description of the parallel between designing mobile robots and RISSC agents. [5] argues that industrial robotic systems with simple sensors and effectors coupled with effective geometric algorithms can be both inexpensive and reliable. Their design philosophy matches ours. We were influenced in defining topology-based navigators and structure detectors by the work of [8, 8, 14] who consider the problem of determining the information requirements to perform robot tasks using the concept of information invariants and perceptual equivalence classes.

- *Knowbots.*

  We are inspired by the recent flurry of activity in the area of designing intelligent task-directed

[1] These systems are quite brittle [25]. In a recent experiment in accessing articles about Clinton's campaign in the Minnesota Daily via Gopher the query "Clinton on health care" resulted in an empty list while the query "Clinton and health care" yielded 19 articles. Computing answers by intersecting word indices is not very robust. The Minnesota Daily does not index the word "on" [25].

agents that live and act within realistic software environments. These have been called knowbots [16] in the context of the national electronic library project. A knowledge-level specification of knowbots is provided in Chapter 4 of [16] which outlines the required capabilities without regard to how they can be realized. The RISSC architecture can be treated as a specific implementation proposal for this specification, organized around the idea of high-level structure. In common with Kahn and Cerf, our initial applications are "in the retrieval of documents for which a user may be able to specify an imprecise description". The RISSC agent architecture can also be used to set up Personal Library Systems[16] "that selectively view, organize, and update contents" of an electronic library for individual use. RISSC agents allow the user to specify queries at a high level of abstraction. The knowledge of the task and environment embedded in the structure detectors and navigators allows for effective traversal of a heterogeneous environment.

- *Information Retrieval and Filtering.* We can interpret the classical work in information retrieval (IR) as follows: the data environment is text, the unit of structure is typically a word, the detectors constructed in the IR literature are pattern-matchers that extract structure at this level, the navigators are index structures over documents. The distinction between the approach proposed here and that of classical IR is that our patterns are at a coarser level: tables, graphs, pictures, *etc.* This permits us to handle heterogeneous forms of information (numbers, weather maps, simulation data). We rely at the most basic level on structure detectors and navigators designed by the IR community [23, 24].

  Work in information filtering (IF) can be viewed as a complement to the work on information retrieval, where the focus of research is in determining precise characterizations of the information in an environment that is relevant to a user. Approaches in the IF community to unobtrusive user modeling in multi-media environments, and calculation of the information that can be ignored based on query specifications are important in the design of long-lived RISSC agents as opposed to transient agents generated in response to one-time queries.

- *Automated document structuring.*

  The goals of the document structuring community are to identify the key constituents of a document image (sections, paragraphs, pictures, *etc.*) from its layout and represent the logical relationship between these constituents. This is a first step towards analyzing the contents of documents.

  Document structuring is usually done in two phases. In the first phase, the location of the blocks on the page is determined. In the second phase, the blocks are classified and the logical layout of the document is calculated.

  Previous work on block segmentation of documents include [13, 21, 29]. The methods are developed in the context of very specific applications; namely, segmenting pages of technical journals [21, 29]and locating address blocks in letters [13]. The methods (the run-length smoothing algorithm [30] and the recursive XY cuts algorithm [21]) use area-based techniques; that is, every pixel in the document is examined at least once in the process of generating the segmentation. We develop a new perimeter tracer block segmentation scheme for any printed electronic document that does not examine pixels in the interior of any identified block. For most documents, this method performs better than area-based schemes.

  Previous work on classifying and logically relating blocks includes [28, 27, 9, 21, 30, 19, 29]. Methods for discriminating between text blocks and pictures in bitmapped images are presented in [29, 9, 28, 21]. A language for representing the hierarchical structure of documents is given in [9]. [28] extracts the logical structure by a bottom-up approach which starts with the finest units of structure and computes aggregates. A similar method is in [19].

2 The Architecture

2.1 The Problem

The *Information Capture and Access* problem is: given an electronic data environment, capture it by acquiring partial models of it and access it guided by tasks posed by users. This problem has been studied in specialized contexts for a long time within computer science. For instance, the field of relational databases has a solution for a special instantiation. Our interest is in solving the problem with performance guarantees for *large unstructured* data environments, with *uncertainty* representation and interpretation.

By "unstructured" we mean that the data has not been preprocessed to fit a particular format. An unstructured environment is not entirely devoid of structure. The readily available forms are at the lowest level, but higher-level structures can be actively acquired. For example, in an electronic journal, we would like to support queries of the form "find the paper that has an alcohol molecule on the top corner of the second page." Here, the data representation is at the character/pixel level, while the retrieval request is phrased at a conceptual level.

By "uncertainty in representation" we mean noise in the basic data representation. For example, the data from a high-energy physics experiment is a sequence of finite-precision numbers which approximate positions of particles along real trajectories. By "uncertainty in interpretation" we mean noise in the process of matching the data and the answer to the user's query. For example, a query to the data generated by a high-energy physics experiment is whether a particle of a given type was produced as a result of a collision. This is done by comparing an ideal curve against the trajectories resulting from a collision. This interpretation process has an approximation component.

The are several reasons why the Information Capture and Access problem is hard in its most general form. There is a mismatch between the granularity of data representation and the answer to a query. The data environment is too large for exhaustively searching it at the lowest level of granularity. There is a need for constructing intermediate goals or partial models that help in the filtering process to bridge the gap between the data and the answer. But these are not intractable problems: even though the data has no overall structure, there is hidden structure at a higher grain that can be detected and used to develop retrieval algorithms. The stock market example in the introduction illustrates this point.

There are some hard and important global questions that arise in this context.

- For a given class of retrieval queries and data environments, what are the appropriate structural abstractions?
- Given a retrieval task, how much intermediate structure must be detected?
- Can these structures be detected reliably and efficiently?
- Can navigators be constructed such that they deliver appropriate information to structure detectors?
- Can we compose an agent out of basic structure detectors and navigators to solve a particular task?
- What are the right communication protocols between structure detectors and navigators in a RISSC agent?
- Given an agent what class of retrieval task is it good for?

We believe that the structure-based approach implemented using robust detectors and navigators is significant for solving the general information capture and access problem. Progress in answering high-level queries in multi-media environments can be made by finding crisp solutions to the questions posed above. This paper does not present general answers to these questions but provides an architecture for the Information Capture and Access problem and specific solutions in the context of two classes of retrieval tasks.

2.2 The Overview

We view RISSC agents as entities that sense and act in the information world. An information access task is solved by constructing a RISSC agent that roams the environment, aided by its navigators, looking for patterns, recognized by its structure detectors. The RISSC approach to Information Capture and Access bears a strong similarity to the search of physical environments by mobile robots. Structure detectors are virtual sensors and navigators are virtual effectors. We are inspired by successful mobile robot architectures in defining the desirable properties of good information capture and access systems.

- **Modularity.** Successful mobile robots are not monolithic units, rather they are assembled out of robust modules. Modularity ensures that by understanding the behavior of the individual pieces and the nature of the communication glue between them, we can effectively predict the behavior of the system as a whole. Modularity is essential for both the analysis and synthesis of agents.
- **Simplicity.** Successful mobile robots are configured from a basic sensor and effector suite with established error properties. In a modular architecture where simple components are connected by simple methods, it is easy to predict the overall behavior and to isolate faulty components when failures occur. This lesson has also been learned in areas other than robotics: namely RISC computing and software engineering. In the information capture and access problem this requirement translates to designing simple structure detectors and navigators with well-characterized performance guarantees.
- **Reliability.** Since our working environments are noisy, like those of mobile robots, structure detectors that look for perfect matches and navigators that rely on perfect data interpretation perform poorly. Consider for instance the design of a structure detector for tables. The general organization of a table is in rows and columns, but the actual table content and layout differ with each example. Tables with partially misaligned or missing records should be detected with a specified degree of accuracy and confidence. We require structure detectors and navigators to be robust in the presence of uncertainty in data representation and should have bounds of accuracy for data interpretation.
- **Efficiency.** The objective of the information capture and access problem is to find good answers in response to a user's query as fast as possible. The data environment is so large that neither the structure detectors nor the navigators can afford to look at every data item, so it is important to design these units efficiently. Good agent design consists not merely of fast structure detectors and navigators. It also requires providing good communication schemes between components. Thus, the natural complexity measure for information agents, as in mobile robotics, should take both communication and computation costs into account.
- **Error detection and recovery.** Since our agents work in a distributed, wide-area network environment, it is important that navigators that physically traverse this environment detect and recover from network failures. The modules in

the system must be self-correcting. This was one of the lessons about organization of complex systems that was [3, 4] discovered in the context of mobile robotics and insect intelligences.

Our challenge is to design a set of simple structure detectors and navigators with provable properties, and provide automated planning schemes for configuring them so that special-purpose information agents can be created quickly and reliably. We believe that large monolithic information retrievers which attempt to be everything to everybody are baroque and difficult to use. Instead, our architecture is based on a library of components with guaranteed performance measures that can be composed in a task-directed manner.

2.3 Navigators

Navigators are programs that plan a traversal of the data for the structure detectors. In other words, they actively make "maps" at the various granularities of representation. Each map is a partial model of the world. In the stock market example, the representation of the newspaper at the paragraph level is such a model. More formally, a navigator n is specified as a function that takes a topology of the world W and produces a refinement of that topology[2].

$$n : \tau_1(W) \rightarrow \tau_2(W)$$

Given a world W, we can construct different models, each suited for a specific set of tasks. The set $\mathcal{T}$ of all possible partial models of W relative to a set $\mathcal{N}$ of navigators is a lattice whose elements are topologies ordered by refinement. The lattice of topologies reveals structures at different levels of detail. The top element $\top$ of $\mathcal{T}$ is the trivial topology of W[3]. The join operation $\vee$ on the lattice takes two topologies and produces a coarsening of both. The meet operation $\wedge$ takes two topologies and produces a refinement of both. The bottom elements $\perp$ are topologies whose open sets are at the base level of data representation. $\mathcal{T}$ is generated by $\mathcal{N}$ as a finite composition of navigators. Composition of navigators is well-defined if they are consecutive segments of a path in $\mathcal{T}$.

The abstract specification of a navigator allows us to prove properties about its behavior that are implementation-independent. Modeling the world as a lattice of topologies provides computational support for the idea of generating successive refinements that zoom-in to the answer. Zooming-in corresponds to traversing the lattice by computing topological refinements.

We can view a navigator as taking a partial model of the world, identifying interesting destinations, and generating route maps for travel between them. For the stock market example discussed in the introduction, a navigator partitions The New York Times into block segments that correspond to paragraphs and incrementally provides them to a table detector. The stock market navigators provide the means for a logical traversal of the world. In contrast, agents like Netfind [26] invoke navigators that physically traverse the Internet looking for information. Here, detecting and recovering from network failures is important and thus we require that navigators have error detection and recovery schemes associated with them.

2.3.1 An Example: Segmenting Documents

Given a pixel array of a document, the goal is to partition the document into regions that capture its layout. This problem has two parts: determining where the regions are and classifying them according to their layout structure. Examples are titles, text blocks, pictures, tables, captions, *etc.* The importance of devising methods that analyze and make use of the logical structure of documents to facilitate information access at levels above keywords has already been recognized by [27, 9, 19, 30, 29].

In this section we present an algorithm for finding the regions of layout structure.

Definition 2.1 *Let B be a polygonal segment of the document. A border of width d is the set $Border(B,d) = B \oplus S^1_{\frac{d}{2}} - B$. Every element of $Border(B,d)$ is a white space*[4].

We restrict our attention to polygonal segments that are *unions* of axis-parallel rectangular regions of the document.

Definition 2.2 *A layout region B relative to a border of width d is the smallest document segment for which $B \cup Border(B,d)$ is entirely contained within the document.*

This parameterized geometric definition (on d, the width of the border) allows us to construct a hierarchical layout model of the document 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. We can thus classify the regions of the document according to the width of 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 $d > 0.2$ inch extract the entire title, while values of $d < 0.1$ inch separate each character of the title into its own block.

Our method for segmenting a document into layout regions requires a set of significant values of d sorted in decreasing order. It takes as input a document represented as a pixel array and produces a tree representing its hierarchical layout. The root of the layout tree is the entire document. The nodes at level i are regions relative to a border of width d_i. The tree is constructed recursively. Level $i+1$ is constructed by finding all the regions of width d_{i+1} inside each node

[2] Let τ_1 and τ_2 be two topologies over a set X. If $\tau_1 \subset \tau_2$ then τ_2 is refinement of τ_1 [20].

[3] The trivial topology of W contains W and $\emptyset$

[4] $A \oplus B = \{a + b \mid a \in A, b \in B\}$ is the Minkowski sum of sets A and B. $S^1_{\frac{d}{2}}$ is a circle of radius $\frac{d}{2}$.

Notation:

$C = \{\langle v, b\rangle\}$ denotes the critical vertex list
where v denotes an (x, y) location in the pixel array
and b is a sweep direction
(1 for horizontal and -1 for vertical).
R denotes the list of vertices for the new regions.
O denotes the list of vertices for the input region.
d denotes the border width.
p is the pixel array for region O.

Input: d, O, p.

Initialization: $C = (\langle first(O), 1\rangle)$.
R = O.

While $C \neq \emptyset$ **do**
begin
$v = first(first(C))$; $b = second(first(C))$;
$pop(C)$
sweep p from v in direction b looking for
runs of white space r longer than d.
For each r that has not been visited along b,
$C = C \cup \{\langle r, -b\rangle\}, R = R \cup \{r\}$.
end

Figure 1: The Perimeter Tracer for the Block Segmentation Algorithm for level i

at level i. Intuitively, each node at level i is processed by rolling a coin of diameter d_{i+1} in its maze of white spaces. We represent a region as a list of vertices obtained by traversing it in clockwise order. Each node at level i is processed as in Figure 2.3.1.

This algorithm essentially traces perimeters of regions. The regions are determined by connecting the vertices found by the perimeter tracing algorithm. This should be contrasted with the existing methods for block segmentation, for example the *run length smoothing* [30] and the *recursive x-y cuts* procedure [21]. These algorithms examine every pixel in the document. Our perimeter tracer has the advantage of examining only the 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 newspaper pages, this is a significant reduction in complexity.

The sizes of the border width have to be chosen carefully for this algorithm to perform well. However, a poor set of border width performs just as well as existing area-based methods. We have three methods for obtaining the set of boundary widths. In the simplest case, the significant ds are provided as input. The significant ds can be computed by randomly sampling regions of the document to obtain the inter-line and inter-character spacing. The values for d can also be acquired by a complete analysis of one large enough region of the document. These methods rely on the fact that most documents are typeset with a distinguished layout using a finite set of fonts. Each font size has a characteristic spacing between the lines and characters. A subset of the generic typesetting rules provided in [21] are assumed:

1. Printed lines are roughly horizontal.
2. The base lines of characters are aligned.
3. Word spaces are larger than character spaces.
4. Paragraphs are separated by wider spaces than lines within a paragraph.
5. Illustrations are confined to rectangular frames (but we have a generalization of the method described here to eliminate this assumption.)

The algorithm, as presented assumes that there is no noise in the data and that we can reliably detect borders. In particular, on any sweep line, horizontal or vertical, we will find lines of length larger than d. In reality we expect a run of white pixels of a certain length to be polluted by black pixels that occur as noise, at random locations. We associate a "black weight" with each coin, which is the number of black pixels that it can ignore. This weight is proportional to the coin size. Metaphorically speaking, each coin is like a bulldozer that can push a certain number of black pixels[5].

The output of this algorithm can be phrased in our topological framework by observing that the union of all the regions at level i of the layout tree constitutes the topology for the document and each level is a topological refinement of the previous level. In fact, the algorithm described in 2.3.1 is the navigator that refines the existing partition of the document at level i to level $i+1$. Each border width corresponds to a navigator, and determining the tree of layout structure is an iteration over all these navigators.

2.4 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. Structure detectors are computable functions 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.

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

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. Recall that $\tau(W)$ is a topological set. Text, numeric and digital audio and video data environments are topological sets, due to their discrete nature.

[5] We thank Jim Davis for this idea.

```
08/22/96 - Titan IV/Centaur Launch
03/29/97 - 66 Maja Asteroid Flyby
06/08/98 - Earth Gravity Assist
02/06/00 - Jupiter Gravity Assist
12/06/02 - Saturn Arrival
03/27/03 - Titan Probe Release
03/29/03 - Orbiter Deflection Maneuver
04/18/03 - Titan Probe Entry
06/30/03 - Iapetus Flyby
05/20/04 - Dione Flyby
09/12/04 - Enceladus Flyby
08/14/05 - Iapetus Flyby
12/31/06 - End of Primary Mission
```

Figure 2: Table of Key Scheduled Dates for the Cassini Mission

2.4.1 An Example: Detecting Tables

To illustrate this definition, consider defining a structure detector to extract all the tables in an electronic document. The world W is the document, and the representation $\tau(W)$ consists of the paragraph blocks that comprise the document. The table detector extracts all the blocks that are tables. The property of being a table is described in terms of vertical alignment of columns and horizontal content consistency [18]. The robustness property is necessary in designing this detector because not all tables have perfectly aligned columns. Our structure detector is built around concise specifications of the misalignment tolerance.

We now outline the design of an algorithm for efficiently recognizing whether a block of text is a table. The input for this algorithm is the result of performing OCR over a topology generated by a block segmentation navigator. Since tables are textual structures at the level of paragraphs, this input is processed by a navigator operating with a border width determined by the inter-paragraph spacing.

The first step in the algorithm is to create the *white space density graph* associated with it, by calculating the percentage of blank spaces in each column of the block.

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 $w(\text{“ ”}) = 1$ and $\forall c \neq \text{“ ”}, w(c) = 0$.

Definition 2.3 *The white space density graph* [18] *of B is the polygonal line* $\text{WDG} : [0, m] \rightarrow [0, 1]$ *defined by the points* $\text{WDG}(i) = \frac{1}{n} \sum_{j=0}^{n} w(B_{i,j}),\ \forall i \in N \cap [0, m]$

.

A column that serves as a separator in a perfect table consists of 100% blank spaces. Column separators in tables with slight irregularities consist of $100\% - \epsilon$ blank spaces. Figure 4 shows the WDG associated with the text in Figure 2. This graph has two perfect peaks, which are associated with the rivers of white space flowing between the three columns in Table 2.

In the second step, the WDG is analyzed to extract its local maxima above $100 - \epsilon\%$. If there are no such maxima, the block of text is not a table. If there are

```
(defun recognize (line-list)
  (let* ((wdg (calculate-wdg line-list))
         (column-separators (find-local-maxima wdg)))
    (if (atom column-separators)
        column-separators
        (lexical-analysis lines column-separators))))
```

Figure 3: Lisp Code for Table Recognition

local maxima, they are candidates for column separators in a possible table and they define the candidate columns. The WDG in Figure 4 has two perfect peaks which suggest three columns.

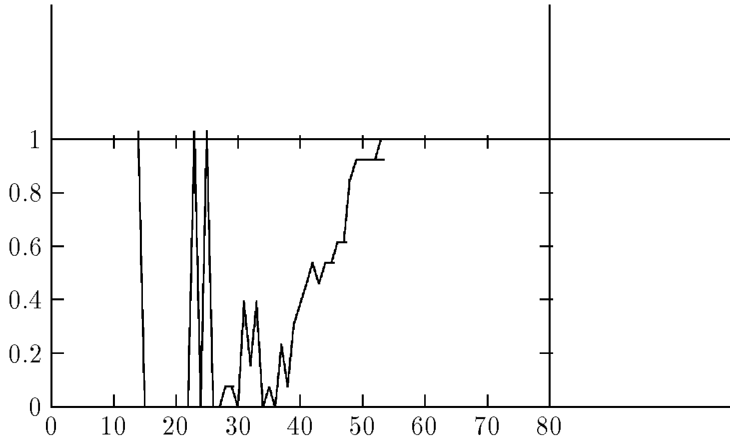

Figure 4: A White Space Density Graph for Table 1

In the third step, each candidate column is analyzed for lexical structure. If there is lexical structure in each column, and if these structures can be combined into lexical structure across the columns for the entire table, the block of text is a table. Otherwise it is not. For Figure 2, the first column is described by the expression $num/num/num$, the second column is described by the expression $-$, and the third column is described by the expression let^*. The three columns are compatible in that the regular expression $num/num/num - let^*$ describes each record of the table. Figure 3 gives the top-level function that implements the table detector.

In [18] we analyze the failure modes of this algorithm. For instance, a stanza of a poem by e. e. cummings would be classified as a table. A block of text will be misidentified as a table if there is an accidental alignment of white spaces. Starting with simple assumptions about English text, we bound the probability of misclassification.

Structure detectors function as filters for navigators. They generate subsets of the topology they take as input for further processing.

2.5 The RISSC Architecture

A RISSC agent computes the answer to a user's query from the given data environment by generating

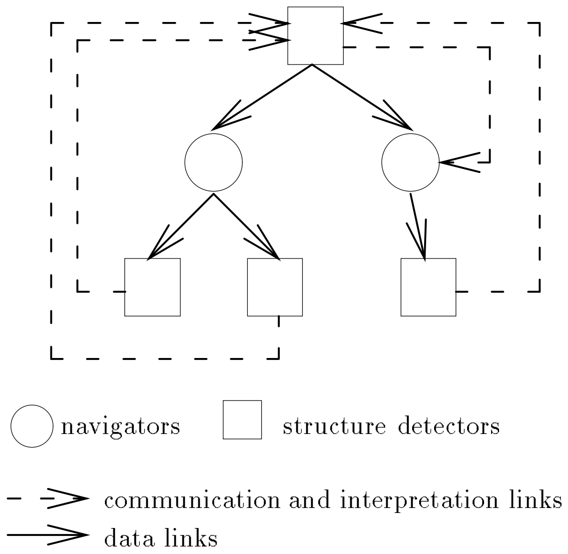

Figure 5: The RISSC architecture for Information Capture and Access

a sequence of increasingly detailed views of smaller and smaller data subsets.

A RISSC agent is a tree of structure detectors and navigators glued together by communication paths, as in Figure 5. The root of the tree is a structure detector for the user's query. We view the query as a request for detecting and retrieving some specific structure from the world. The tree consists of alternating layers of structure detectors and navigators. Arcs between the nodes are communication paths. There are two types of communication paths. The first type flows downwards through the tree, connecting structure detectors and navigators. Structure detectors convey subsets of the world to the navigators below them. Navigators send a topology to the structure detectors below them. In addition to these data paths, a RISSC 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.

A RISSC agent is created in response to a user's query. We envision automated planning systems that synthesize agents using libraries of structure detectors and navigators from given task specifications. The focus of this paper is not on automated synthesis, but on presenting a framework for information capture and access and illustrating it with hand-assembled RISSC agents. In a future paper, we will describe task-directed planners for generating agents.

Consider again the problem of determining the evolution of the $AT\&T$ stock price over the past year using electronically archived copies of Fortune magazine. Stock price information can be found in tabular as well as graphical form in this data environment. At the highest level, a RISSC agent for this task invokes a navigator for segmenting the document into paragraph blocks. The output of this navigator is passed to a table detector and a graph detector. The tables collected by the table detector are given to a navigator that extracts its records. This information is further processed by a string comparator that detects the record for $AT\&T$. Similarly, the graphs recognized by the graph detector are partitioned into graphical objects and text, which are then processed by string comparators.

We now discuss how the answer to the user's query is assembled. The string comparators, which are structure detectors at the lowest level, send the results of their computations as they are found, to the table and graph detectors. These, in turn, send the information to the root node which assembles the stock report in the form of a stock price for each business day of the year. There are three scenarios for the answer at the root: correct, incomplete, and inconsistent.

1. For each business day of the year there is at least one number and when there are two numbers, they are identical.

2. For some business days of the year there are no numbers.

3. For some business days of the year there are two conflicting numbers.

The incomplete error mode can be handled by dynamically reconfiguring the RISSC agent to spawn a new path in the tree to look for the missing data. To handle the inconsistent error mode, a simple policy is to prefer the values returned by the more accurate detector. Since detectors have associated error bounds that measure their reliability and robustness, this scheme is easy to implement.

All the answer-assembly computations, including error detection and recovery, rely on communication protocols. We are in the process of developing a library of communication protocols for agent design. We have identified the need for: (1) synchronization primitives to coordinate computations along different paths in the tree; (2) agreement primitives, for answer assembly; and (3) termination primitives, to abort obsolete computations. We show examples of their use in section 3.

The computational support for this architecture is in a library of structure detectors and navigators. Each component has well analyzed behavior and known error bounds. Each structure detector solves a special instance of the information capture and access problem reliably. Each structure detector and navigator pair (s_i, n_i) solves a class q_i of queries within specified performance bounds. We can compose two structure detector and navigator pairs (s_i, n_i) over q_i and (s_j, n_j) over q_j to yield a new structure detector and navigator pair (s_{ij}, n_{ij}) that solves queries in $q_i \cup q_j$. If the structure detector s_i and s_j recognize properties p_i and p_j respectively, then s_{ij} recognizes the logical disjunction p_i or p_j. Additionally, if the error bounds for s_i and s_j are ϵ_i and ϵ_j then the error bound for s_{ij} is $max(\epsilon_i, \epsilon_j)$.

This composition operation is not useful for synthesizing an entire path in the tree, which is the ultimate goal of automated agent design. We view this as a very challenging and important problem and hope that this paper will inspire work in formalizing other notions of composition that could support the development of automated planning methods for designing special purpose but complex agents.

Information capture and access systems are being developed to aid humans to sift through large volumes of information quickly. The structure-based methods we propose can be viewed as computational models of the human ability to find information based on layout cues. The zoom-in, zoom-out strategy we propose emulates the process of a human flipping through the pages of a book to locate information with some visual pattern.

3 Examples

3.1 Assembling Stock Reports

In this section we describe the design of the agent that solves the stock market problem posed in the Introduction. More data from the actual implementation is shown in [18].

The input for this problem is a set of scanned issues of The New York Times. The RISSC agent for this task is a shown in figure 6. 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 picture using methods discussed in [29]. The remaining blocks are partitioned at the character level, by the block segmentation algorithm operating with inter-character spacing as the border width. The output of this navigator is processed by a character recognizer, that produces ASCII paragraph blocks[6]. To maintain the alternating structure of the tree, the ASCII paragraphs generated by the character recognizer are passed through the identity navigator to the table detector described in Section 2.4.1. The subset of paragraph blocks that are recognized as tables are provided to a row navigator, in order to separate the records. The rows are individually passed to a detector which is a string comparator, to identify the *AT&T* records. The row navigator also passes the rows to a header detector that filters the row corresponding to a header, if it exists.

There are communication and interpretation links between the string comparator and the table detector, the table detector and the root node, and the root node and the character recognizer.

The link between the string comparator and the table detector is used for passing and interpreting the *AT&T* record. If the user's query had been to report all information about *AT&T*, no interpretation would be needed at the level. In fact, the entire record could be passed up from the table detector to the root and reported to the user. However, the task is more specific and requires reporting only the closing values. The field for the closing value has to be identified and extracted from the record retrieved by the string comparator. The interpreter on the link between the string comparator and the table detector uses knowledge about the layout of stock tables. If there is a header, the interpreter scans it looking for the keyword "close", and determines the location of the closing value in the record. If there is no header, the interpreter uses other information about the format of the table. It uses the fact that stock market tables usually contain three columns corresponding to high, low, and closing values, and that the closing value is between high and low. This information, which can be encoded as rules, can be used to identify the location of the closing column.

The link between the table detector and the root is used to communicate the closing value.

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 often 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.

3.2 Locating Bibliographic References Through the Internet

In this section we describe the design of an agent that takes a partial description of a bibliographic reference in the form of information about some fields: title, author, institution, subject, address, location *etc.* It returns a complete bibliographic reference or the document representing that reference. As an example, consider the user query "retrieve the complete reference on the article on assumption-based truth maintenance written by deKleer sometime in the 1980s". This query does not provide any information about the institute affiliation of deKleer and only gives a partial description of the contents and date. The agent navigates through the Internet looking for ways to complete the partial bibliographic reference using, for instance, the bibliography databases stored at Stanford, to produce the following:

```
@article{Kleer86a,
  author =      "Johan de Kleer",
  title =       "An Assumption-based {TMS}",
  journal =     "Artificial Intelligence",
  year =        "1986",
  volume =      "{\bf 28}",
```

[6] This navigator-detector pair is an OCR system.

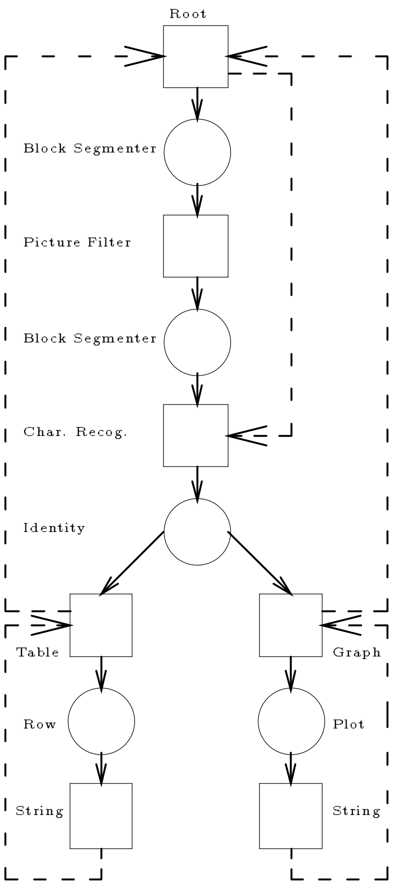

Figure 6: A RISSC agent for compiling stock reports.

```
pages =        "127-162",
publisher =    "North-Holland"}
```

As another example, consider the query "retrieve recent articles on machine learning and artificial intelligence 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 (in general paths, at the various sites) that contain articles on machine learning. The data environment for this query is heterogeneous and distributed. While there are information servers that can retrieve and present information given its location and form, our task is to locate a partially specified document in a very large, poorly structured environment, and recognize and retrieve it in a reasonable amount of time. We have implemented the following RISSC agent to accomplish this task.

The query goes to the root node that invokes as navigator the Unix utility `cd`. Using the Alex file system [6] this agent can travel through the Internet using anonymous `ftp` in a transparent manner, by simply executing `cd`. `cd` sends the agent to a remote directory in the Internet. 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. The navigator, structure detector combination is repeated until actual files are reached. String comparators are then used to select files of interest to the user. 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. This is a simple process, unlike the previous example, where sophisticated answer assembly was required. The complexity of this agent arises from the need for having considerable knowledge about the Unix file organization in each structure detector in the tree. We illustrate this using an example from our existing implementation. The query is to find recent technical reports on machine learning. When the agent for this task reaches the directory `/vol/alex/edu/mit/ai` it has a choice of seven large subdirectories to search. Our detector uses the README file, or prompts the user to select a subset of subdirectories for further investigation.

In our previous example of assembling stock reports the information required by the user was fully specified: namely a list of closing $AT\&T$ stock values for each day in a specified interval of time. The challenge was to develop an agent that retrieves this information from the provided sources in a correct, robust, and economical way. In this problem however, the user's query is only partially specified, The goal of the agent is to find information to complete the user's query using the Internet as a resource. While in our previous problem all navigators were logical, the navigators here are physical and we need to account for network failures.

Our agent handles a finite set of network errors by using structure detectors for recognizing error messages from anonymous ftp.

This agent can also retrieve and play audio and video documents that are stored in recognizable formats across the Internet. The only change is in the structure detector at the leaves of the agent tree.

3.3 Other Examples

1. Automated Weather Map Interpreter

   This environment consists of a database of images with time-stamps associated with them, describing weather patterns. The types of queries we would like to answer are "given a weather map with a specific configuration of a moving warm and cold front, is a tornado likely to occur?" To answer this query, we scan the database for a prior instance of the same configuration. Note that this necessitates detecting a structure at a higher level of granularity than the data representation.

   The highest level navigator partitions the data into sequences that describe the same event. Tracking a cold front involves determining its starting and ending frame in the image sequence.

The structure detectors are surface matchers that identify the regions of the image that correspond to the objects of interest, *i.e.* cold fronts. In addition, they combine the detected location with the direction of motion to filter with high probability the irrelevant sequences. This operation can be used to build indices over such data.

2. Robust Internet locators

   The environment is the Internet and the problem involves locating information across it. One class of problems is finding a complete email address of an individual based on geographic information. Netfind [26] is one solution to this problem. Netfind is a seed-based strategy that requires detailed location specifications. Queries of the form "find the email address of Gerry Salton who is at a north-eastern university" are not handled. To get an answer the user has to manually step through all the north-eastern states, and then through each university in the state.

   Netfind can be viewed as a structure detector and navigator pair that does not partition the internet addresses adequately to support incomplete queries. We can make up for this deficiency of Netfind by constructing an agent using `whois`, `finger`, `netfind`, `WAIS`, and `Gopher`-based access to on-line phone books. Each tool works with a different partition of the Internet.

   Examples of other queries that can be answered similarly are "find a nearby library with a particular journal" and "find a public domain copy of TeX"

3. Scanning Video Tapes

   The environment is a video tape and the problem involves automatically finding the beginning of a sequence with some layout characteristics. An example query is "find the next sequence with the camera focused on the white board."

   A navigator partitions the frame into related sequences. Recent tracking technology can support this [12]. These sequences are then passed to structure detectors that can quickly match geometric shapes [11].

4 Conclusions

We have presented a methodology for building simple RISSC agents to respond to conceptual queries in multi-media environments by taking advantage of the underlying structure of information. We proposed the construction of simple structure detectors and navigators glued together by communication, as a paradigm for structuring the search for the answer into a number of manageable data layers. We have identified a number of issues that need to resolved for constructing RISSC agents from query specifications. Our solution increases the flexibility and range of queries that can be supported in a rich data environment. This simplicity in the underlying components is offset by the need for developing effective communication protocols, error-detection and recovery schemes, and answer assembly algorithms. This is the most challenging part of designing a RISSC agent. The RISSC agents that were presented in this paper can be schematized for solving a range of similar problems.

The design philosophy advocated in this paper is diametrically opposite to that behind large, general-purpose systems like WAIS, Archie, and Gopher. We believe that our library-based scheme is suited for fast prototyping of special-purpose information agents with performance guarantees. By way of instantiation, we present a detailed analysis of an agent that extracts stock market information and an agent that retrieves articles from the Internet. When is this design methodology more appropriate than others? To answer this question, we need a theoretical basis for characterizing information capture and access tasks and measuring the effectiveness of alternate architectures on these task classes. The notion of structure introduced in this paper is a first step in this direction. Structure is an information invariant which characterizes a class of tasks in an implementation-independent manner. More experimentation in building RISSC agents is needed to gain additional insight into scaling issues.

Acknowledgements

We thank John Hopcroft for proposing the problem of information capture and access, and for his guidance and support. Thanks go to Rich Zippel for suggesting the weather map interpretation problem, and to James Allan for careful reading of an early draft. Thanks also to Jim Davis, Bruce Donald, Dean Krafft, Denise Moore, T.V. Raman, Jonathan Rees, Matthew Scott, Kristen Summers, and Rich Zippel for enlightening discussions.

The first author has been supported by the Advanced Research Projects Agency of the Defense Department under ONR Contract N00014-92-J-1989, by ONR Contract N00014-92-J-39, and by NSF Contract IRI-9006137. The second author has been supported by NSF Contract IRI-8902721.

This work was supported in part by the Advanced Research Projects Agency under Grant No. MDA972-92-J-1029 with the Corporation for National Research Initiatives (CNRI). Its content does not necessarily reflect the position or the policy of the Government or CNRI, and no official endorsement should be inferred.

References

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

[2] 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.

[3] R. Brooks, Elephants don't play chess, *Design of Autonomous Agents*, ed. P. Maes, MIT/Elsevier, 1990.

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

[5] J. Canny and K. Goldberg, A "RISC" Paradigm for Industrial Robotics, to appear, *Proceedings of the International Conference on Robotics and Automation*, 1993.

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

[7] P. Crean, C. Russell, and M. V. Dellon, Overview and Programming Guide to the Mind Image Management Systems, Xerox Technical Report X9000627, 1991.

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

[8] B. Donald and J. Jennings, Constructive recognizability for task-directed robot programming, *Journal of Robotics and Autonomous Systems*, 9(1), 1992.

[9] H. Fujisawa, Y. Nakano, and K. Kurino, Segmentation methods for character recognition: from segmentation to document structure analysis, *Proceedings of the IEEE*, vol. 80, no. 7, 1992.

[10] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[11] D. Huttenlocher, G. Klanderman, and W. Rucklidge, Comparing images using the Hausdorff distance, to appear, *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[12] D. Huttenlocher, J. Noh, and W. Rucklidge, Tracking non-rigid objects in complex scenes, Cornell University Technical Report TR92-1320, 1992.

[13] A. Jain and S. Bhattacharjee, Address block location on envelopes using Gabor filters, *Pattern Recognition*, vol. 25, no. 12, 1992.

[14] J. Jennings and D. Rus, Active model acquisition for near-sensorless manipulation with mobile robots, in *Proceedings of the IASTED Conference on Robotics and Automation*, 1993.

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

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

[17] M. Lesk, The CORE electronic chemistry library, Proceedings of the SIGIR, 1991.

[18] C. Lewis and D. Rus, RISSC Informatics – Part 2: Robust table recognition and extraction, forthcoming technical report, Cornell University.

[19] M. Mizuno, Y. Tsuji, T. Tanaka, H. Tanaka, M. Iwashita, and T. Temma, Document recognition system with layout structure generator, *NEC Research and Development*, vol. 32, no. 3, 1991.

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

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

[22] S. Robertson, The methodology of information retrieval experiment, *Information Retrieval Experiment*, in K. Sparck Jones, Ed., pp 9-31, Butterworths, 1981.

[23] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.

[24] G. Salton and C. Buckley, Improving retrieval performance by relevance feedback, *Journal of American Society for Information Science*, vol. 41(4), pp. 288-297, 1990.

[25] G. Salton, *Automatic Text Processing: the transformation, analysis, and retrieval of information by computer*, Addison-Wesley, 1989.

[26] M. Schwartz and P. Tsirigotis, Experience with a Semantically Cognizant Internet White Pages Directory Tool, *Journal of Internetworking Research and Experience*, March 1991.

[27] Y. Tanosaki, K. Suzuki, K. Kikuchi, and M. Kurihara, A logical structure analysis system for documents, *Proceedings of the second international symposium on interoperable information systems*, 1988.

[28] S. Tsujimoto and H. Asada, Major components of a complete text reading system, in *Proceedings of the IEEE*, vol. 80, no. 7, 1992.

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

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