Customized Information Extraction as a Basis for Resource Discovery

Darren R. Hardy
Michael F. Schwartz

Technical Report CU-CS-707-94
March, 1994

Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430
(303) 492-3902
schwartz@cs.colorado.edu

Abstract

Indexing file contents is a powerful means of helping users locate documents, software, and other types of data among large repositories. In environments that contain many different types of data, content indexing requires type-specific processing to extract information effectively. In this paper we present a model for type-specific, user-customizable information extraction, and a system implementation called *Essence*. This software structure allows users to associate specialized extraction methods with ordinary files, providing the illusion of an object-oriented file system that encapsulates specialized indexing methods within files. By exploiting semantics of common file types, Essence generates compact yet representative file summaries that can be used to improve both browsing and indexing in resource discovery systems. Essence can extract information from most of the types of files found in common file systems, including files with nested structure (such as compressed ''tar'' files). Essence interoperates with the Wide Area Information Servers (WAIS) system, allowing WAIS users to take advantage of the Essence information extraction methods.

1. Introduction

Publishing tools like Gopher and World Wide Web are contributing to a dramatic increase in the volume of Internet-accessible information. Yet, it is increasingly difficult to locate relevant information among this burgeoning sea of data. Browsing a hierarchy containing millions of directories is infeasible, particularly given the erratic organization that results when the data are created by many different people. Instead, automated search procedures are needed to help users locate relevant information. For efficiency reasons, this *resource discovery* problem requires indexing the available information. In turn, building an effective index requires indexing methods tailored to each specific environment.

As an example, consider the problem of locating information about networking protocols among the global set of Gopher data. A simple solution would be to index the menu names. Yet, this would not work well for standards documents named according to committee designations (such as X.25). Indexing the contents of Gopher documents would provide better support, but would produce much larger indexes and would require data type-specific indexing procedures. In particular, one would not use the same techniques to index both textual documents and binary executable programs (e.g., to allow networking software to be located). Furthermore, it would be useful to be able to tell an indexing tool what files *not* to index. For example, it may be unnecessary to index multiple versions of revision-controlled documents. Avoiding extraneous files can make the index more compact, and the results of searches less cluttered.

The points illustrated by this example can be summarized by one observation: Different applications and different execution environments require customized means of extracting and summarizing relevant information to support resource discovery. Ideally, all data would be represented as objects, with attached indexing methods. Because a good deal of information currently exists in non-object oriented repositories (like UNIX[1] files), we have developed a system that creates the illusion of typed objects for ordinary, unencapsulated files. To do this, we provide a framework for recognizing file types and applying type specific selection and extraction methods to files.

This paper makes two contributions. First, we decompose the information extraction problem into customizable components that allow a single system to be used in many different situations. Second, we present measurements of a tuned, well-exercised system implementation, which incorporates an understanding of UNIX file semantics and context to generate compact yet representative summaries for general collections of file data. We call our system *Essence* because of its ability to reduce large amounts of data to relatively small content summaries. Essence supports a more flexible collection of data extraction mechanisms than any related system, can handle more of the types of files found in common settings, and achieves much more space efficient content summaries than these systems. Essence exports summary data through the search and retrieval interface of the Wide Area Information Servers (WAIS) system. During the past year, people in 1,060 sites in 45 countries have used the WAIS interface to search Essence summary information.

We present an overview of related resource discovery tools in Section 2. In Section 3 we describe the Essence information extraction model. We discuss the user view of the system (including the WAIS interface) in Section 4. We discuss details of our implementation in Section 5, and measurements in Section 6. We discuss future work in Section 7, and offer our conclusions in Section 8.

2. Overview of Related Resource Discovery Systems

Most existing Internet information systems export and retrieve data, providing only per-server indexing support. The oldest such system is *anonymous FTP*, a convention that allows users to transfer files to and from machines on which they do not have accounts [Postel & Reynolds 1985]. Prospero lets users organize FTP and other files according to personal preferences by creating *views* that can span sites, based on several naming features [Neuman 1992]. The Internet Gopher has more recently become a widely popular means of publishing information on the Internet [McCahill 1992], through a client/server protocol that lets users browse and retrieve data from a variety of different repositories. The World Wide Web (WWW) has also become quite popular, supporting hypertext documents based on access method-independent links that can point to documents in many different information spaces. Starting in 1993 Mosaic became an exceedingly popular WWW client

[1] UNIX is a registered trademark of UNIX System Laboratories.

[Andreessen 1993], because of its attractive graphical interface and ease of use for accessing many types of data reachable via WWW links.

WAIS lets users index and export mostly textual information on servers that can be queried across the Internet [Kahle & Medlar 1991]. WAIS generates indexes containing every word that appears in a textual document. For certain other types of data (e.g., various bibliographic database formats), WAIS extracts keywords based on knowledge of the particular document type. WAIS divides its indexes among the servers that provide information, and provides a single global index of small descriptions of each registered WAIS database in the Internet.

A number of systems have been developed to provide network-wide indexes of information exported by one or more of the above publishing tools. The first global indexing tool was Archie, which periodically gathers anonymous FTP directory listings from each of approximately 1,100 UNIX FTP archives world-wide [Emtage & Deutsch 1992]. Because Archie indexes only file names, a single index can hold information about many resources. However, the index only supports name-based (as opposed to content-based) queries. The Veronica system provides similar functionality, but indexes Gopher menus rather than FTP file names [Foster 1992]. The WHOIS++ service gathers templates describing information content and administrative information about each participating site [Weider, Fullton & Spero 1992].

A final class of related work involves tools that exploit knowledge of file structure to extract and index information. The first system to use this technique was the MIT Semantic File System (SFS) [Gifford et al. 1991]. SFS uses file naming conventions and file contents to determine file types, and then runs type-specific *transducers* to extract index keywords. SFS provides a *virtual directory* interface, interpreting directory names as queries against the index, and providing query results as files in virtual directories. Essence and SFS use similar techniques for extracting keywords. However, SFS targets its mechanisms to the file system abstraction. Essence provides a more general mechanism, because it is not tied to any particular file system or search interface. Also, while the directory abstraction is familiar to users, SFS's reuse of this abstraction interacts with the semantics of certain common operations. For example, if a user tries to copy data into a virtual directory (created as a result of an SFS query), a special case must be invoked that treats the operation as a permission denial. Beyond these differences, Essence supports more of the file types found in common file systems, and generates more compact summaries than SFS does.

The Nebula system also gathers keywords from file system data, but uses an explicit typing mechanism rather than heuristic techniques for recognizing files [Bowman et al. 1994]. Nebula uses a set of grammars to extract the data. Essence allows arbitrary programs to extract data, including programs that use grammars to describe the extraction algorithms.

3. Information Extraction as a Basis for Resource Discovery

Essence decomposes the information extraction problem into components that are independent of how the data are stored or exported. Decoupling the extraction process from the storage mechanism and export interface provides a flexible substrate on top of which to build resource discovery systems, allowing the extraction mechanisms to be tuned without changing the rest of the system. Well tuned extraction methods can improve browsing by reducing the amount of information that users must peruse. Tuning can also make extracted data more compact by excluding unimportant keywords, such as common programming language constructs in source code files. A well-tuned extraction engine can also make searching more precise, by biasing the weights of keywords that are known to be important (such as those extracted from document titles and author lists).

The main premise behind Essence is that information extraction is most effective when exploiting the semantics of particular types of files and particular execution environments. For example, by recognizing a file as a document from a particular word processing system, Essence can apply knowledge of that system's syntax to locate the document's authors, title, and abstract. Moreover, by recognizing that a file falls within a narrowly used part of a file system (based on local organizational conventions), Essence can avoid extracting the information altogether.

To exploit these types of semantics, Essence breaks information extraction into the four steps illustrated in Figure 1. The *type recognition* step uses various methods to determine a file's type. While performing this step, Essence sometimes encounters files that are encoded according to some presentation layer [Tanenbaum 1988] format (such as compression or multi-file ''bundling''). The *presentation unnesting* step transforms such files into an unnested format. Each file's name and typing information are then examined by the *candidate selection*

step, to select which objects are to be summarized. Finally, the *summarizing* step applies a type-specific extraction procedure to each selected object.

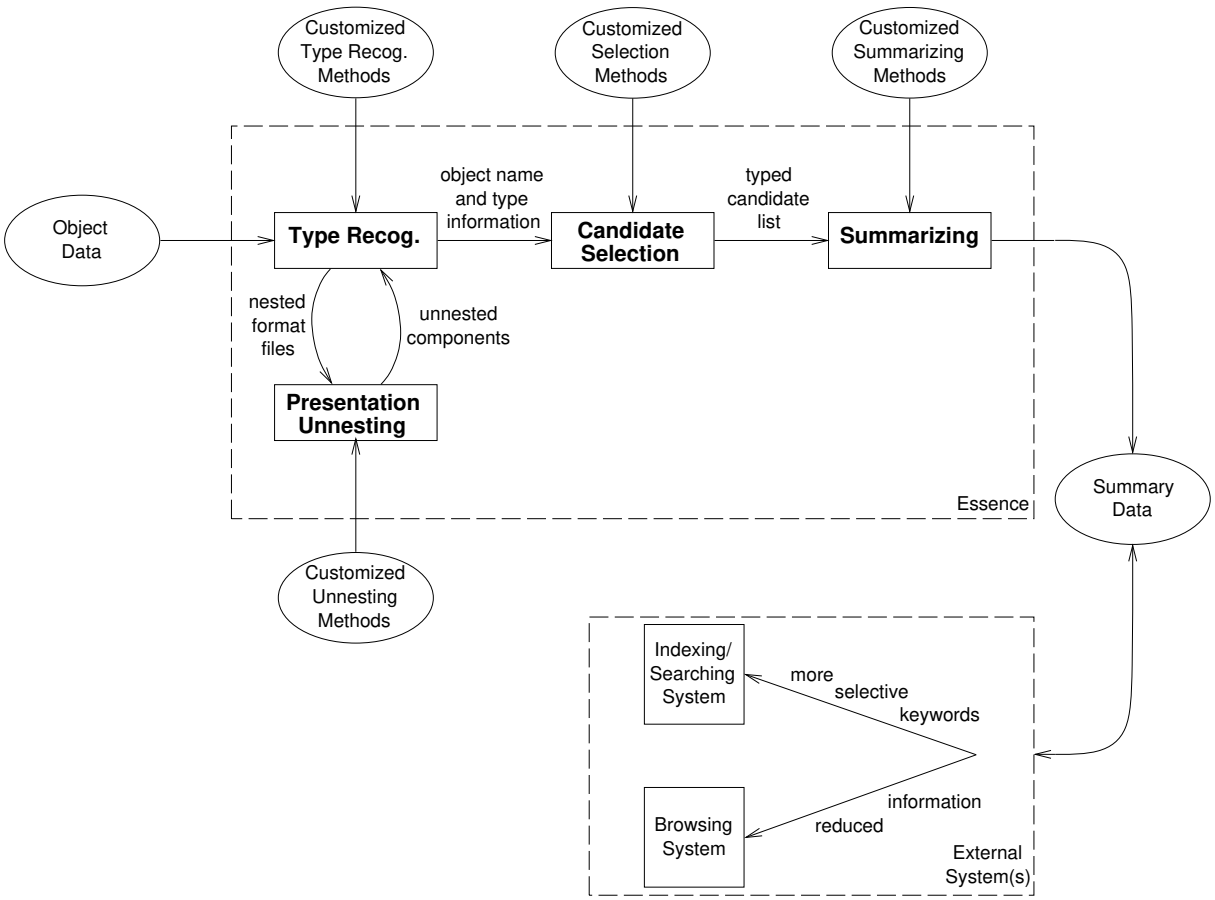

Figure 1: Information Extraction Steps

Each of the steps in Figure 1 can be customized, although our implementation provides a set of appropriate defaults. We consider the steps in more detail below.

Type Recognition

In an explicitly typed file system (such as a PC containing OLE [Microsoft Inc. 1993] documents), the type recognition step can simply extract type information for encountered object from the file system itself. For untyped file systems (such as UNIX), type recognition must use a variety of heuristics to recognize files. For example, this step can exploit file naming conventions (e.g., noting that PostScript files are often named with the extension ‘‘.ps’’). Implicit typing can also be done by inspecting data within a file (e.g., noting that PostScript files begin with the string ‘‘%!’’).

Presentation Unnesting

Along the way to typing files, the type recognition step sometimes encounters files encoded according to one or more presentation layer data transformations. These transformations arise because of heterogeneity and other complexities in a distributed environment, such as compression or ASCII-encoding. Table 1 describes a variety of presentation formats that arise in common file system environments.

Unlike the *types* expected by the candidate selection step, presentation *formats* are semantics-independent, and hence imply nothing about how best to select or summarize data. For the purposes of information

Format	Typical Purpose	Example Software
Compression	Storage and transmission efficiency	UNIX *compress* [Welch 1984] GNU *gzip*
Encryption	Privacy and security	UNIX *crypt* [NBS 1977] PEM content encryption [Kent 1993]
Standardized data representation	Accommodating heterogeneous hardware-level data representations	Sun XDR [Sun 1987] ISO ASN.1 [ISO 1987]
ASCII-encoding	Transmitting binary data via electronic mail	UNIX *uuencode* Macintosh *BinHex*
File ‘‘bundling’’	Grouping related files for distribution	UNIX *tar* UNIX *shar* PC *ZIP* Macintosh *StuffIt*
	Grouping related executable library code for link editing	UNIX *ar*
Compound document formats	Structured, type-encapsulated documents	MIME [Borenstein & Freed 1993] OLE [Microsoft Inc. 1993]

Table 1: Popular Presentation Nesting Formats

extraction, presentation layer transformations must simply be ‘‘unraveled’’ to expose the underlying files, whose types *do* imply semantics that can be exploited during candidate selection and summarizing.

When a presentation-nested file is encountered, it is unnested into one or more result files. The result files themselves can also be nested. For example, uuencoded, compressed tar files are commonly used for transmitting entire UNIX directories through electronic mail. As illustrated in Figure 1, Essence handles this case by iterating the type recognition and presentation unnesting steps. The final result of this iteration is a set of typed objects.

In addition to unnesting the input files, the presentation unnesting step also keeps a record of the nested origin of each unnested file, for use by the candidate selection and summarizing steps. For example, after the constituent files have been unnested from a directory, the candidate selector can note that the files came from a directory that contained both the source (‘‘.c’’) and object (‘‘.o’’) files for a program, and choose to exclude the object files. The summarizer can use the file-nesting linkage information to decide that an entire directory should be summarized as one unit, so that searches against any of the extracted keywords will match the whole directory, rather than one file in the directory. Doing so reduces the amount of result ‘‘clutter’’ at search time.

Presentation-nested files are prevalent in anonymous FTP file systems, multimedia documents, and PC word processing documents. While nested FTP files arise mainly because of deficiencies in FTP for transmitting entire directories, compound documents are a useful and powerful paradigm. They represent a step towards object-oriented data encapsulation. We believe extracting information from presentation-nested files will be increasingly important in the future.

Candidate Selection

Given a set of typed objects, the candidate selection step chooses objects to summarize. This step allows the system to remove files that would contribute unnecessary information to an index, cluttering the results of searches. Candidate selection is similar to the notion of *stop lists* introduced in the information retrieval literature [Salton 1986]. However, our model allows for more general selection conditions, such as regular expressions for filtering names and types. Our model also allows more complex selection procedures, that attempt to eliminate redundancy among related files. For example, we have implemented a selector that attempts to eliminate object code that can be derived from available source code files. In such a case, the candidate selector gives preference to types that can be more effectively summarized (source code in this case).

Pruning based on name can be useful for eliminating certain files known to contain unneeded information, such as old versions of files signified by extensions ending in ‘‘.bak’’. Pruning based on type can eliminate files for which summarizing procedures are not capable of extracting much useful information. This is most commonly the case for files whose types were not recognized during the type recognition step.[2]

Another aspect of tuning candidate selection is customizing the choices based on the environment being indexed. For example, anonymous FTP archives typically contain popular documents and software packages, which exhibit heavy sharing [Danzig, Hall & Schwartz 1993]. In contrast, general-purpose file systems typically contain mostly user-specific data that exhibit relatively little sharing [Muntz & Honeyman 1992, Ousterhout et al. 1985]. The candidate selection procedures can prune the name space based on such source-specific selection criteria. This is important, because summarizing and indexing narrowly interesting data will mean that search results are ‘‘cluttered’’ by uninteresting matches.

Summarizing

The summarizing step applies an appropriate extraction procedure (called a *summarizer*) to each selected object, based on the type information uncovered in the type recognition step. For example, a summarizer for UNIX manual pages understands the troff syntax of these documents, as well as the conventions used to describe UNIX programs. It uses this understanding to extract summary information, such as the title of a program, related programs and files, the author(s) of the program, and a brief description of the program.

The summarizing step also extracts some information independent of file type, including owner, group, and full file name. Moreover, it lets users add keywords manually if desired. These type-independent processing steps allow some indexable information to be included even for files of unrecognized type.

Summarizers can extract keyword information from both textual and binary files. For example, many binary executables have related textual documents describing their usage, from which keyword information can be extracted. The current implementation only extracts textual keywords found within the binary files, but the model would permit more complex summarizers that, for example, used speech recognition algorithms to index audio data.

If the type recognizer provides relationships between file types, summarizers can share code with one another using an multiple inheritance scheme. For example, our implementation recognizes UNIX manual pages as a special case of troff formatting source, and inherits extraction support from the formatting source summarizer.

Example File Processing

Figure 2 illustrates an example set of steps that a compressed ‘‘tar’’ file might go through on the way to being summarized. After unnesting the compression and tar formats, a set of extracted files are typed and passed to the Candidate Selection step. Recognizing that both the source (‘‘Essence.ms’’) and formatted (‘‘Essence.ps’’) versions of a paper are available, the selector extracts information only from the source version, since that version contains more effectively summarized information. In the example the source code is also excluded from indexing. Finally, keywords are extracted from the selected files, based on knowledge of the semantics of each file type and local site preferences about what data should be extracted from each file type.

[2] It may still be useful to ‘‘summarize’’ unrecognized files using a type-independent summarizing mechanism – for example, extracting the file’s name.

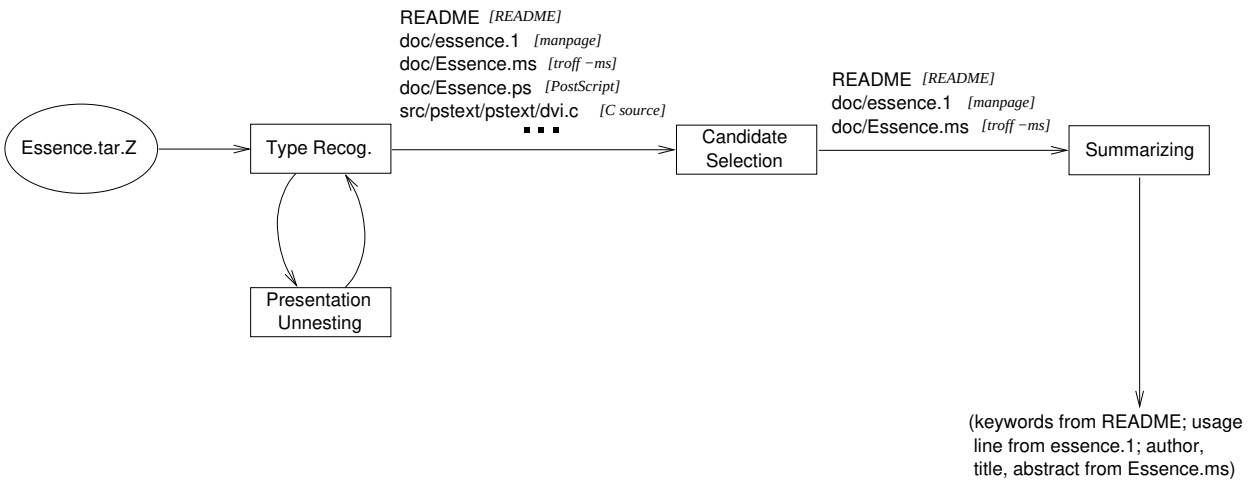

Figure 2: Example File Processing

4. User View of System

To extract information with Essence, the server administrator specifies a list of files and directories to be summarized. Essence recursively descends into any directories that were specified, and passes each file through the type recognition, presentation layer unnesting, candidate selection, and summarizing steps. Essence indexes the summary files using a version of WAIS modified to understand the Essence summary file format, so that it generates meaningful WAIS *headlines*. These headlines provide users with short descriptions of each file, usually a file name. With Essence, headlines include a file's name, the name of the file from which it was extracted (in case of presentation nesting), and the file's type.

Figure 3 shows an example search of an index generated by Essence of the ftp.cs.colorado.edu anonymous FTP file system, searched using the UNIX *xwais* client. The output contains a WAIS rank-ordered list of the top 11 files that best match the search "resource discovery". This figure provides an intuitive illustration of Essence's effectiveness. The most highly ranked match is a PostScript paper that provides an overview of the Internet Resource Discovery Project at the University of Colorado. The second match is a file that contains a brief project overview plus the project bibliography. The third match is a resource discovery thesis done by one of the current paper's authors. The fourth, sixth, and ninth matches were papers about particular resource discovery systems developed at the University of Colorado. The eleventh match was a source distribution for another resource discovery system we developed. The other three matches were other resource discovery papers. These files occur in various formats (text, compressed text, compressed PostScript, and compressed tar). Note also that the data type could be used in queries - e.g., to locate resource discovery SourceDistributions.

In WAIS, a user retrieves files by selecting a matching headline. With Essence, if the headline represents a component nested within a file (such as the last headline in Figure 3), the summary file is retrieved, instead of retrieving the presentation-nested file itself. If the headline represents a plain file (such as the fourth headline in Figure 3), the file is retrieved. This functionality lets users browse remote file systems by retrieving and viewing small summary files before deciding to retrieve complete files. This is useful when interacting across a slow network link.

5. Implementation

The Essence implementation focuses primarily on files found in the UNIX file system environment. Using Essence, a system administrator can summarize and index most of the files found in common departmental and Internet file systems, and export these data via the WAIS search and retrieval interface.

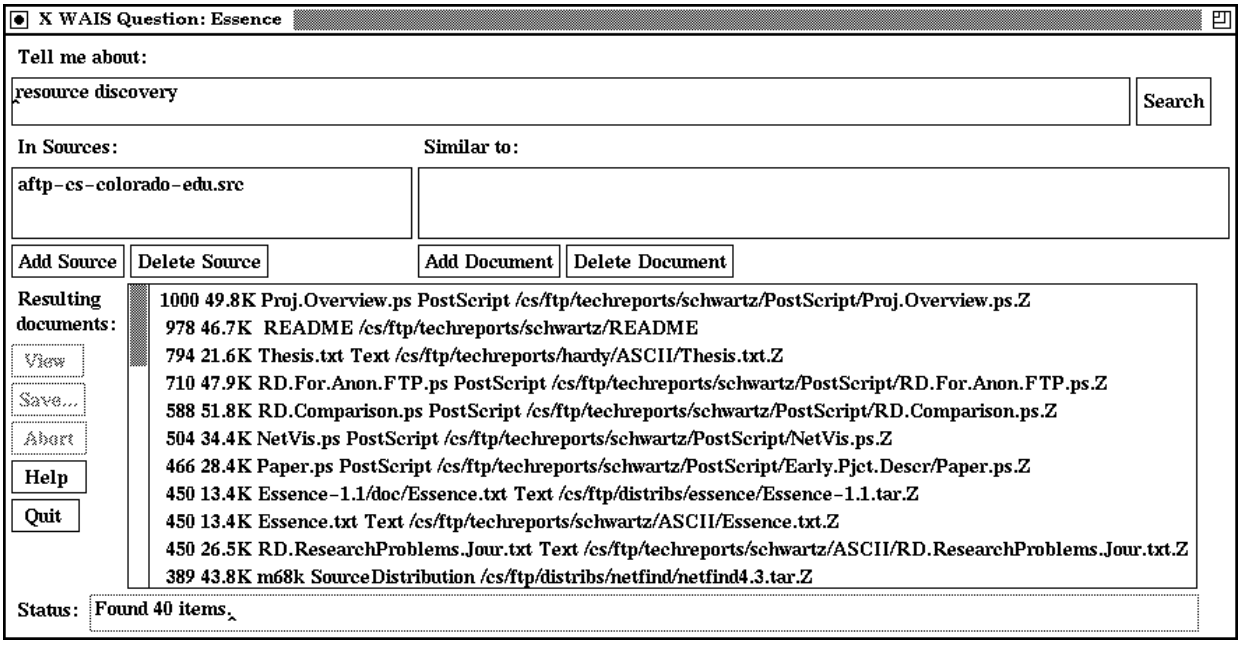

Figure 3: Example WAIS Search of Essence-Extracted File Data

To gain experience with Essence, we made it available both in source form and as an Internet-accessible service. From January to July of 1993, users from 257 sites retrieved the source code, and users from 540 sites used the WAIS interface to the Essence server, generating approximately 5,000 searches. During this time, we received feedback from a variety of users, concerning both performance and customization needs.

Below we describe the implementation of the steps outlined in Section 3, including numerous performance optimizations and other improvements we made in response to users' suggestions and our own experiences with the system.

5.1. Type Recognition

Essence recognizes file types using a combination of file naming conventions, content testing, and user-defined methods. It can also use explicit file typing information, for environments that contain such information.

Naming conventions often correctly identify file types. For example, file names with a ''.ps'' extension are typically PostScript files, while files whose names contain the string ''README'' typically hold information about an entire directory or source distribution. Figure 4 shows some entries from the Essence configuration file that specifies naming conventions for type recognition. In this file, the first field is the file type, and the second field is a regular expression for the corresponding file naming convention.

```
Compressed                      .*\.Z
Makefile                        .*[mM]akefile.*
PostScript                      .*\.(ps|eps)
Revision Control System (RCS)   .*,v
TarArchive                      .*\.tar
```

Figure 4: Example Configuration File for Naming Convention-Based Type Recognition

In addition to using naming conventions, Essence examines file contents to try to determine file types. In particular, many UNIX files contain an identifying number (called a *magic number*) at the beginning of the file . For example, Sun SPARC binary executables start with the hexadecimal number *0x8103*, while Sun Pixrect images start with the hexadecimal number *0x59a66a95*. Furthermore, particular strings within a file may indicate the file type. For example, PostScript images start with the string ‘‘%!’’, while electronic mail files contain lines that begin with header field strings such as ‘‘From:’’ and ‘‘To:’’.

Essence uses the UNIX *file* command [USENIX Association 1986] to support this form of type recognition. As with exploiting file naming conventions, locating identifying data is a rule-based technique, expressed with regular expressions. To modify the rules, users modify the *file* command’s */etc/magic* configuration file. Figure 5 shows some sample entries from this file, where the first field is the byte offset of the identifying data, the second field is the type of this field, the third field is the identifying data itself, and the last field is a description of the corresponding file type.

```
0   string   /*              C program text
0   string   \037\235        Compressed data
0   long     0x8103          Sun SPARC binary executable
0   string   #!.*/bin/perl   Perl program
0   string   %!              PostScript image
```

Figure 5: Example Configuration File for Data Content-Based Type Recognition

Creating a suitable */etc/magic* file is not trivial, because the identifying data must be distinctive. For example, the ‘‘/*’’ delimiter for C programming language comments is not sufficiently distinctive, and will likely appear in a variety of types of files. A lack of distinctive identifying data is common for binary formats, which usually depend on a single magic number. We built the Essence */etc/magic* file through experimentation.

Some file types are difficult to identify using only simple naming conventions and content testing. Therefore, Essence lets users provide their own customized stand-alone programs to identify file types. These programs can use a variety of techniques. For example, we implemented a program that uses simple heuristics to identify software distribution directories (noting that they contain C source code, Makefiles, and README files), for use with one of the candidate selectors we implemented.

As an optimization, Essence applies file type recognition methods in order of expense: naming conventions followed (if necessary) by the ‘‘file’’ command followed (if necessary) by customized recognizer programs. Also, Essence caches file information from the *stat* system call, for reuse in the presentation layer unnesting step.

5.2. Presentation Layer Unnesting

Table 2 indicates the seven presentation unnesting formats that Essence currently supports, along with the corresponding unnesting methods. As an optimization, Essence unnests certain common combinations (such as compressed tar files) as a single ‘‘pipelined’’ step. This avoids the disk I/O overhead of creating intermediate files. As a second optimization, we perform presentation unnesting, candidate selection, and summarizing in a single pass for the *Archive* file format. Doing so reduces disk I/O and processing costs. This optimization is possible because extracting and combining the symbol tables from the unnested object files is equivalent to extracting the symbol table directly from the archive file. We are able to exploit this fact because of the use of customized information extraction.

Nesting Format	Unnesting Description
Archive	Extract archived binary relocatable object files
Directory	Extract individual files
Compressed	Uncompress
GNUCompressed	Uncompress GNU-format file
ShellArchive	Extract contained files
Tar	Extract archived files
Uuencoded	Decode ASCII-encoded contents

Table 2: Essence Unnesting Techniques

5.3. Candidate Selection

Essence lets users prune the set of summarized files based on lists of file types as well as regular expressions for filtering based on file name. Candidate selection can also perform more sophisticated processing. For example, if a set of files was unnested from a single directory that contained only software and documentation files, the candidate selector recognizes the collection as a *SourceDistribution*. In this case, it invokes special rules for excluding extraneous files – for example, excluding object and revision control (RCS) files that have corresponding source files.

While candidate selection is conceptually one step, for performance reasons it is broken into two parts. Some files are rejected based on a name-based stop list before type recognition, avoiding the overhead of identifying a file's type. Files that pass this filter are then typed and passed through a second filter, which selects files based on their types.

5.4. Summarizing

Essence's summarizers are stand-alone UNIX programs that are easy to write and integrate into the system. Each summarizer is associated with a specific file type and understands the type well enough to extract summary information from the file. Currently, Essence supports summarizers for twenty-five file types, as listed in Table 3.

The processing for these types fits into a multiple inheritance hierarchy, corresponding to the type structure illustrated in Figure 6. All types inherit type-independent processing steps, which include the file's name owner, and group, and also let users add keywords manually. The text formatting summarizers inherit code to deal with textual extraction. The source distribution summarizer makes the most involved use of the hierarchy, inheriting methods from the textual extractors, source code and Makefile extractors, and ManPage extractor. As discussed below, the binary extractor inherits methods from the ManPage extractor when it cross-correlates an executable program with its manual page.

The following subsections provide more detail about the techniques used in some of the summarizers, representative of Essence's supported file types.

File Type	Summarizer Description
Audio	Extract file name
Bibliographic	Extract author and titles
Binary	Extract meaningful strings and manual page summary
C, CHeader	Extract procedure names, included file names, and comments
Dvi	Invoke the RawText summarizer on extracted ASCII text
FAQ, README	Extract all words in file
Font	Extract comments
Mail	Extract certain header fields
Makefile	Extract comments and target names
ManPage	Extract synopsis, author, title, etc., based on ‘‘-man’’ macros
News	Extract certain header fields
Object	Extract symbol table
Patch	Extract patched file names
Perl	Extract procedure names and comments
PostScript	Invoke the RawText summarizer on extracted ASCII text
RawText	Extract first 100 lines plus first sentence of each remaining paragraph
RCS	Extract RCS-supplied summary
ShellScript	Extract comments
SourceDistribution	Extract full text of README file and comments from Makefile and source code files, and summarize any manual pages
SymbolicLink	Extract file name, owner, and date created
Tex	Invoke the RawText summarizer on extracted ASCII text
Troff	Extract author, title, etc., based on ‘‘-man’’, ‘‘-ms’’, ‘‘-me’’ macro packages, or extract section headers and topic sentences.
Unrecognized	Extract file name, owner, and date created.

Table 3: Essence Summarizer Techniques

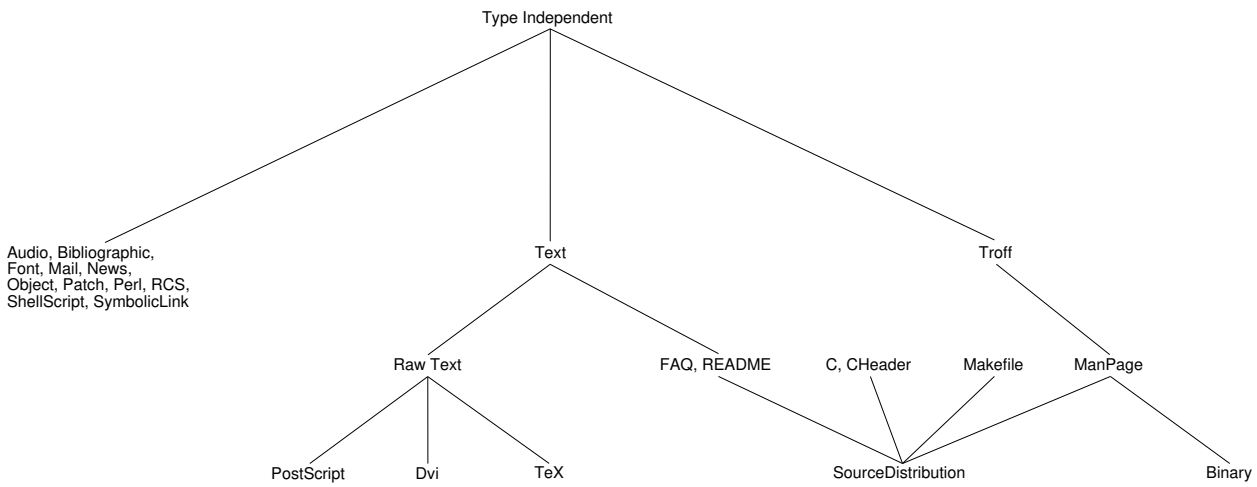

Figure 6: Multiple-Inheritance Type Hierarchy Basis for Summarizer Code Sharing

Binary

An obvious method for a *Binary* summarizer is to extract ASCII strings from the binary file, using the UNIX *strings* command. Because these strings typically contain a good deal of unimportant information, Essence filters them using heuristics that only retain strings that convey the program's purpose – such as usage, version, or copyright information. Essence also cross-references binary executables with associated manual pages, and generates keywords using the *ManPage* summarizer.

Raw Text, FAQ, and README Files

Raw text is difficult to summarize because it is unstructured. Rather than attempting a complex natural language processing approach, Essence assumes that the most useful keywords in raw text files lie near the beginnings of files. This heuristic works well, for example, with paper abstracts or tables of contents. For this purpose, the *RawText* summarizer extracts all of the words from the first one hundred lines of each file. Because many users of our initial prototype reported that this technique did not capture enough of the content of raw text files, we added the capability to extract the first sentence of every paragraph after the first one hundred lines.

Some unstructured text, like FAQ (Frequently Asked Questions) and README files, typically contain relevant, concise information about a software distribution or application. The *README* summarizer extracts every word from its input files, typically providing useful keywords without consuming too much space.

Text Formatting Source

Although text formatting source (such as TEX, Troff, or Word Perfect) is more structured than raw text, effectively summarizing these files is difficult unless semantic information is also available. For example, plain Troff files or Troff files using the ''-me'' macro package are difficult to exploit semantically, because their syntax is associated with formatting commands (such as font size or line spacing), rather than more conceptual commands (such as macros to indicate an author's name or a paper's title). Troff files using the ''-ms'' or ''-man'' macros are much easier to summarize, because they use such conceptual macros.

Essence supports a sophisticated summarizer for Troff using any of the ''-me'', ''-ms'', and ''-man'' macro packages. The *TeX* summarizer only extracts ASCII text from TEX (or LaTEX) files using the UNIX *detex* program. Exploiting TEX semantics would be a straightforward extension of the methods used in our *Troff* summarizer.

The *Dvi* and *PostScript* summarizers extract keywords by first converting the file to its corresponding text, and then running the *RawText* summarizer on the extracted ASCII text. In our original implementation [Hardy & Schwartz 1993], dvi files were first converted to PostScript using *dvips*, and then processed using the *PostScript* processor. After experimentation, we changed this summarizer to use the *dvitty* program instead, because it does a better job of extracting the text from the page positioning codes. We also adopted an improved *PostScript* summarizer, which extracts text in different ways depending on what typesetting system generated the PostScript. This significantly reduced the occurrence of cases where words were split apart, and represents another example of the advantages of customized extraction.

Source and Object Code

Both source and object code are highly structured, and contain easily exploited semantic information. The *C* summarizer extracts procedure names, header file names, and comments from a C source code file. The *Object* summarizer extracts the symbol table from an object file.

Some extra semantics can be exploited in the case where source code is included in part of a directory recognized as a *SourceDistribution*. First, the summarizer uses the file-nesting linkage information recorded at presentation unnesting time to decide that the entire directory should be summarized as one unit. Second, the summarizer attempts to extract copyright information in the source or object files. This information typically contains project, application, or author names. The summarizer also extracts keyword information from README files, because these files often contain useful information about the directory's contents.

Other Possible Summarizers

Many other potential summarizers are possible. For example, Lisp or Pascal summarizers could be implemented with straightforward extensions to the existing source code summarizers. In contrast, audio or image summarizers would be difficult to implement, because summarizers are currently limited to keywords. If they were not limited to keywords, one possibility would be to sample a bitmap file down to an icon. While this would not easily support indexing, it could be used to support quick browsing before retrieving an entire image across a slow network link, similar to what is done by the Dienst system [Davis 1994].

6. Evaluation

In this section we evaluate four aspects of the Essence implementation: keyword quality, space and time efficiency, the overall costs of each of the model steps, and the completeness of the supported presentation unnesting and summarizer methods.

6.1. Keyword Quality

An important issue in the evaluation of any keyword-based discovery system is the quality of keywords that it supports. The usual approach to measuring keyword quality is to compute *precision* and *recall* values for a set of user queries, against a given set of documents (called a *reference set*). Intuitively, recall is the probability that all of the relevant documents will be retrieved, while precision is the probability that all of the retrieved documents will be relevant [Blair & Maron 1985]. More precisely,

$$Precision = \frac{Number\ of\ Relevant\ \text{and}\ Retrieved\ Documents}{Total\ Number\ of\ Retrieved\ Documents}$$

$$Recall = \frac{Number\ of\ Relevant\ \text{and}\ Retrieved\ Documents}{Total\ Number\ of\ Relevant\ Documents}$$

Obtaining meaningful precision and recall measurements is difficult, because they require manual analysis of what constitutes the relevant set of documents in response to a particular set of keywords. Moreover, because the notion of relevance is subjective, to be truly representative this analysis would have to be repeated separately for each of a large number of users. Doing so becomes impractical with large reference sets.

Because of these difficulties, we took a different approach to measuring keyword quality for Essence. Observing that WAIS has become a widely accepted ‘‘industry standard’’ for information retrieval, we chose to measure how closely Essence could retrieve exactly the documents that the WAIS logs indicated were relevant, for each logged WAIS search. In other words, we measured precision and recall *relative* to WAIS.

We began with logs for a popular WAIS database containing three years of full-text CACM articles, handled by a server run by Thinking Machines, Inc. WAIS logs a great deal of information, from which we extracted the keywords used for each search, the list of matching documents, which documents users retrieved,[3] and the list of documents used as relevance feedback input.[4] For each search we defined the *Relevant Document Set (RDS)* to be the set of documents that the user either retrieved or selected for relevance feedback (based on the CACM WAIS search logs). We then ran each of the logged searches against an Essence-based index, and computed precision and recall values based on this RDS. More precisely, for each search we make the following definitions:

[3] When a WAIS match occurs, it is presented to the user as a headline. Users must explicitly select documents to be retrieved, based on these headlines.

[4] The WAIS relevance feedback mechanism lets users select a particular set of documents from the results of a query for use in recomputing weights for future queries – in effect, asking the system to locate other documents ‘‘like these documents’’.

$$\textit{WAIS Result Set } (WRS) = \textit{the set of documents matched by WAIS}$$

$$\textit{Essence Result Set } (ERS) = \textit{the set of documents matched by Essence}$$

$$\textit{Relevant and Retrieved WAIS Set } (RRWS) = WRS \cap RDS$$

$$\textit{Relevant and Retrieved Essence Set } (RRES) = ERS \cap RDS$$

Finally, we define the relative precision and recall for Essence as follows:

$$\textit{Relative Essence Precision} = \frac{|RRES|}{|ERS|} \Big/ \frac{|RRWS|}{|WRS|}$$

$$\textit{Relative Essence Recall} = \frac{|RRES|}{|RRWS|}$$

Note that this approach does not provide (or require) precision and recall measurements for WAIS itself. It simply measures how closely Essence is able to match all and only the documents that WAIS users indicated were relevant. Moreover, these measurements provide a lower bound on Essence's recall, as it is likely that the logged WAIS searches did not locate some documents that users would have deemed relevant and that Essence would locate.[5] The current measurement experiment highlights only the times when Essence was unable to locate documents WAIS located, but not vice versa. Moreover, because Essence can extract data from various binary file types that WAIS does not understand, there are cases in which Essence would achieve higher recall than WAIS. Also, because Essence is easily customizable, its precision and recall can be improved for specific environments. Finally, we note that in a full system, Essence would likely be used to generate multiple indexes at varying levels of detail, ranging from full-text indexes at ''leaf'' information servers to more terse summaries at higher level servers [Bowman et al. 1993].

Figure 7 presents the results of this experiment, for a range of result set sizes. These results were based on a set of 653 searches for which users either retrieved documents or specified relevance feedback, between October 1 and November 30, 1993.

As Figure 7 shows, Essence's precision approaches that of WAIS as the result set sizes. We did not attempt a larger size than 256 because users are unlikely to browse the result headlines for large result set sizes. (small) user-defined size,[6] so Essence does not have much leeway to return fewer documents than WAIS.

[5] Although WAIS provides full-text indexing, in most cases result set size limitations cause the returned documents to constitute only an essentially random subset of the actual matching documents. Therefore, it is possible that Essence can return a relevant document that WAIS missed.

[6] 40 is the default size using the xwais client.

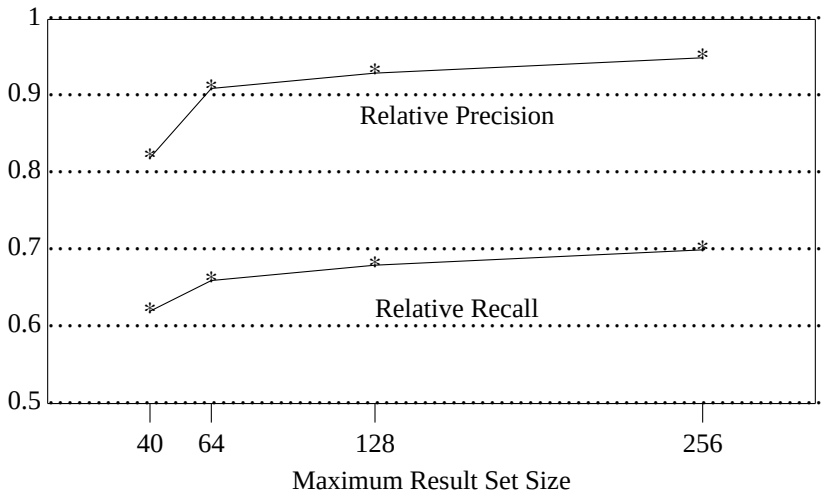

Figure 7: Relative Precision and Precision Measurements for Essence

Intuitively, relative recall should be less than 1 because Essence omits some of the keywords that WAIS includes in the index. In a few cases relative recall could have been improved if the CACM articles had more structure. This is because the missing keywords appeared in portions that Essence ignored, such as the reference lists. Intuitively, one would also expect relative precision to be greater than 1, because Essence matches fewer documents per search. Essence's relative precision is less than 1 because the server truncates result sets to a

6.2. Space and Time Efficiency

Because it only extracts keywords from selected parts of documents, Essence can generate more compact indexes than a full-text system like WAIS, and can generate these indexes more quickly as well. For example, Essence's *RawText* summarizer skips much of the data. In this section we evaluate the space and time efficiency gains made possible by this approach. Space efficiency is important, because a single space-efficient index can represent information about many resources, and hence support far-reaching distributed searches. Time efficiency is less of an issue because information extraction can be parallelized or performed incrementally. Even so, time efficient extraction is useful, for example for supporting on-the-fly extraction and searching [Manber & Wu 1994].

Because Essence separates data extraction from indexing but SFS and WAIS combine these steps, we compare the cost of Essence extraction plus WAIS indexing with the cost of SFS and stand-alone WAIS. To do this, we measured space and time requirements of each system for each file type supported by Essence, and weighted these measurements according to typical file type occurrence frequencies and sizes. For this purpose we measured *indexing rate* as the speed of extracting and indexing the data, and *summarizing ratio* as the size of the source data divided by the size of the generated index. This latter measure highlights the space savings provided by selective indexing versus full text indexing. To obtain realistic mixes of file types, we performed these measurements in two common environments: a departmental file system that contains commonly shared data and tools in the University of Colorado Computer Science Department, and the department's anonymous FTP file system. We chose these two environments because they typify two different common uses of file systems. Finally, we computed aggregate time and space measurements using the formula:

$$\sum_{i=1}^{n} (f_i a_i) v_i ,$$

where f_i is the frequency associated with file type i, a_i is the average file size associated with file type i (in KB), v_i is the indexing rate or summarizing ratio associated with file type i, and n is the number of non-presentation-nested file types supported by the system. $f_i a_i$ is used to normalize the measurements, to reflect only each system's supported file types, excluding presentation-nested files. (We discuss the costs of presentation unnesting in Section 6.3.)

The results of these measurements are summarized in Table 4. These measurements were performed on a Sun 4/280 server running SunOS 4.1.1, with local SMD disks and 64 megabytes of main memory.

File System	Indexing or Summarizing Rate (KB/min)		Summarizing Ratio (source size / index size)	
	Essence	**WAIS**	**Essence**	**WAIS**
Dept. Shared File System	2,589.67	1,422.89	38.81	1.16
Anonymous FTP Server	1,961.69	2,073.01	21.93	2.33

Table 4: Weighted Time and Space Averages Based on File Type Frequencies

Essence achieves faster indexing rates for file types where the overhead of customized information extraction is overshadowed by the cost savings of running WAIS indexing against fewer input keywords. For example, this is true for the *RawText* summarizer, because it excludes many words from the indexing step.

While detailed measurements were not available, using numbers from [Gifford et al. 1991] we estimate that SFS processes data at 712 KB/min. However, because the SFS measurements were performed on a Microvax-3 (which is approximately one-third as fast as the Sun 4/280), SFS's indexing speed appears to be comparable to that of Essence.

The summarizing ratio measurements given in Table 4 show that Essence summaries are only 3-11% as large as WAIS indexes for the file types it supports, in the measured file systems. Again using numbers from [Gifford et al. 1991], we estimate SFS's summarizing ratio to be 6.8.

Filename-based indexers achieve much higher summarizing ratios than those reported above. For example, Archie's summarizing ratio is 765 [Emtage & Deutsch 1992]. The tradeoff, of course, is that filename-based indexes support less powerful searches.

We can compute some simple measures of content capturing efficiency by combining the measurements from this and the previous section. In particular, if we multiply relative precision or recall by the summarizing ratio for each system, we measure how much precision or recall is attained per unit of index expansion factor. Table 5 shows the measurements for the anonymous FTP file system. These figures show that Essence attains much more precision and recall per byte of index than WAIS does.

	Relative Precision * Summarizing Ratio	Relative Recall * Summarizing Ratio
Essence	20.83	15.35
WAIS	2.33	2.33

Table 5: Content Capturing Efficiency

6.3. Model Component Costs

Table 6 indicates overall costs for the implementation of each of the model steps illustrated in Figure 1. These measurements were performed on the departmental anonymous FTP data, on a DECstation 5000/125 running Ultrix 4.2, with 32MB of RAM and a local SCSI disk. Presentation unnesting is quite costly because of its inherent I/O intensity, and the need to touch file data once for every unnesting operation. Interestingly, these

costs roughly mirror the corresponding presentation layer costs of data transmission, which are predicted to become the dominant cost of networking in this decade [Clark & Tennenhouse 1990].

Model Step	% Time
Type Recognition	23.8
Presentation Unnesting	36.6
Candidate Selection	1.1
Summarizing	38.5
Total	100.0

Table 6: Model Component Costs

Much of the current system is implemented as interpreted scripts (in awk, sed or Perl). This choice supports easy customization and rapid prototyping, but also impacts performance. Process creation also affects performance, because many of the system components execute as subprocesses (e.g., forking the *file* command to classify files, and various programs to summarize files). As an example, to summarize and index the anonymous FTP file system in our measurements, approximately 15% of the time was spent forking UNIX processes and loading executable images. This overhead could be reduced by moving to a single shared address space environment. We believe the overhead is warranted, given the low ratio of indexing vs. search operations in most resource discovery systems, and the ease that users have of creating new customized components as separate scripts. WAIS took the opposite choice, and the need to modify the source for each new file type is a frequently cited critique of the system.

The fact that Essence reduces file data to such small summaries means indexing costs are also quite small. For example, using the Table 6 workload we found that, of the the total time to run Essence plus WAIS indexing, only 3% of the time was spent indexing.

Table 7 shows how much space overhead Essence incurs when unnesting presentation-nested files in the measured anonymous FTP file system. In this table, *Original Data* refers to the data that reside in the anonymous FTP file system. *Processed Data* refers to the data that Essence processes while summarizing the *Original Data*, including all of the original (nested) files plus each file extracted during unnesting. *Summarized Data* refers to the data on which summarizers are run – i.e., all of the ‘‘bottom level’’ files after presentation unnesting. There are many more processed than summarized data files because of the prevalence of nested file formats in the measured data. For example, a compressed PostScript file is counted as both a compressed file and a PostScript file for the count of processed data files, but only as a PostScript file in the count of summarized data files. *Summary Data* refers to the resulting summary files.

	Total Number of Files	Total Size (in MB)
Original Data	996	92.58
Processed Data	14,348	431.69
Summarized Data	3,152	261.76
Summary Data	3,152	9.79
Index	2	10.47

Table 7: Presentation Unnesting Space Measurements

The summarizing ratio between original data and index size (92.58/10.47 = 8.8) understates the actual space reduction, because the indexed data actually consumed 261.76 MB. In particular, systems like WAIS that do not support nested structure would have to leave the data unnested. Hence, we actually achieve a twofold space reduction. WAIS would need to keep the unnested data around, and then would generate an index whose size

was comparable to that of the unnested data. Essence generates a smaller index, and can operate on nested data. Combining the measurements, WAIS would require approximately 374 MB of space for the unnested data plus the index (112 MB, using the 2.33 ratio from Table 4) while Essence requires only 113 MB total (92.58 + 9.79 + 10.47). This represents a 70% space savings over WAIS. Clearly, it is worthwhile to support presentation unnesting. WAIS could benefit, for example, by modifications that at least let it work with compressed files. While it currently allows compressed files to be retrieved, it cannot index compressed files.[7]

WAIS requires the index to reside on the same machine as the indexed data. Essence does not have this restriction. In fact, we are currently adopting Essence for use in a resource discovery system that will extract keywords from the source data repositories and send them to a remote indexing server [Bowman et al. 1993]. Separating the index from the source data in this fashion will make Essence's space savings much more pronounced. For example, using the above measurements, Essence would require only 9.79 + 10.47 = 20.26 MB at the remote index server, a 94.6% savings over WAIS. This suggests that we could index a much larger distributed collection of data with Essence than is possible with WAIS.

6.4. Completeness of Presentation Unnesting and Summarizing Methods

Table 8 reports the percentage of data in the measured file systems that Essence, WAIS, and SFS were successfully able to interpret and index. These measurements reflect the fact that 85% of the files in the anonymous FTP file system had nested structure, while only 2% percent of the files in the shared departmental file system had nested structure. The prevalence of nested files in the anonymous FTP file system caused WAIS and SFS to fare poorly in that environment. While the departmental file system contained relatively few nested files, it contained many specialized file formats that the systems were unable to interpret.

	Essence	**WAIS**		**SFS**	
		All Files	Non-Nested Files	All Files	Non-Nested Files
Anonymous FTP Server	97.50	10.40	69.31	11.39	75.94
Dept. Shared File Server	71.15	39.11	39.91	67.99	69.38

Table 8: Percentage of Interpretable Data for Essence, WAIS, and SFS

Independent of presentation layer nesting, Essence supports more of the file types found in common departmental file systems and anonymous FTP file systems than either WAIS or SFS. Although WAIS and SFS support most of the frequently occurring file types (such as *RawText* and *CHeader*), Essence supports the file types that contribute most to overall data size (such as *Archive* and *Binary*). WAIS and SFS each support various types of files that Essence does not support. Examples include MedLine and New York Times formats. There are 8 such formats understood by WAIS, and 14 understood by SFS.

Essence, WAIS, and SFS each support about 30 (partially overlapping) types of files. Among the three systems, 52 different file types are supported. 17 of these types are supported by all three systems, while 23 are supported by a single system. This large overlap and the need for custom file type support argues for a single extensible system like Essence. This would avoid the need to reimplement extraction procedures for each system, and would permit easy sharing of extraction code between the systems. Moreover, Essence is more flexible than these other systems. For example, rather than automatically recognizing file types, WAIS depends on users' explicitly specifying types for each set of files they index. Moreover, WAIS does not support any notion of partial text extraction – it either includes all keywords in a textual document or just a file name for non-textual documents.

[7] The "freeWAIS" system being supported by the Clearinghouse for Networked Information Discovery and Retrieval can index and retrieve compressed files.

7. Future Directions

We plan a number of improvements to Essence, as part of our Internet Research Task Force efforts to support scalable resource discovery [Bowman et al. 1993]. To support increasing information volume, we are extending Essence to use a hierarchical indexing scheme, where higher-level indexes are more widely replicated and searched. Essence will separate indexing information into various degrees of abstraction, so that more important information will be passed up to higher level indexes. For example, in examining a document, keywords extracted from title lines might be more significant than keywords extracted from the abstract and author lines, which in turn might be higher-level than text from the entire document. At a still higher level, summarized documents will be classified according to taxonomy subject terms and associated aliases.

One of the principal types of user feedback we received was the need for higher performance while summarizing and indexing data. By changing Essence to use an in-memory presentation-unnesting mechanism, we can reduce overhead associated with intermediate disk copies. Also, we are adapting Essence to support incremental gathering, and gathering from remote sites. Finally, we will move to a different indexing mechanism, because WAIS indexes are space inefficient. Approximately 50% of their content consists of null characters, and they do not use any compression mechanisms. Instead, will use an indexing mechanism based on Manber's Glimpse system [Manber & Wu 1994].

Another disadvantage of WAIS indexes is that they do not preserve file structure information. We are extending Essence to retain structure information extracted from documents (e.g., the fact that a keyword came from an author macro in a Troff -ms document), and building an indexer that can support structured queries. We are also extending Essence to tag indexed data with information about the source and revision history of data, to aid in tracking down problems, and to support a notion of data quality.

As a longer term goal, we will extend Essence to support more complex types of data, such as records from a relational database. Doing so will require extensions to the index data structures, as well as generalizations of the Essence extraction procedures. Our approach will be to allow users to attach programs to file system directories or database schema, to tell Essence how to extract data from those types of data repositories. Doing this will support a more object-oriented indexing model.

8. Conclusions

Content indexing provides a powerful means of helping users locate relevant information among large repositories. To be most effective, content indexing must exploit the semantics of the different types of data and different execution environments in which it is used. In this paper we presented a model for customized information extraction and an implementation of this model in the Essence system. Essence allows users to associate specialized extraction methods with ordinary files, providing the illusion of an object-oriented file system that encapsulates specialized indexing methods within files.

Essence supports a more flexible collection of mechanisms for recognizing, selecting, and extracting information than either WAIS or SFS. Moreover, because Essence supports presentation unnesting, it can extract information from many more files than either WAIS or SFS for various common settings, such as anonymous FTP archives, CD-ROM collections, and local source trees.

Essence achieves much more space efficient content summaries than either WAIS or SFS, yet manages to capture most of the important keywords from summarized files. In particular, Essence achieves nearly the same precision and 70% the recall of WAIS, but requires only 3-11% as much index space and 70% as much summarizing and indexing time as WAIS. Moreover, it is much easier to write and integrate new Essence's summarizers than to make the corresponding modifications to WAIS, which requires modifying and recompiling a large piece of software. Essence also provides a more general information extraction mechanism than SFS, because it can be used with any storage system or export interface. SFS works only with a virtual directory file system abstraction.

More generally, Essence's support for customized information extraction could form the foundation for a wide range of resource discovery applications, from local file system search tools to wide area distributed archives and databases.

Software Availability

Essence consists of approximately 8,800 lines of mostly C and Perl code, including a number of summarizers adopted with minor modifications from other software packages. The software is available by anonymous FTP from ftp.cs.colorado.edu in /pub/cs/distribs/essence. It is also being incorporated into the ''Free WAIS'' distribution, by the Clearinghouse for Networked Information Discovery and Retrieval.

Acknowledgements

This paper is based on work reported in part in an earlier conference paper [Hardy & Schwartz 1993].

Panagiotis Tsirigotis implemented the source code distribution recognizer and the improved PostScript parsing program discussed in Sections 3 and 5.

Mic Bowman, Peter Danzig, Jim Guyton, Jim O'Toole, and David Wood provided helpful comments on this paper.

We thank Jonathan Goldman for providing us with the WAIS logs needed to compute the measurements given in Section 6.1.

This material is based upon work supported in part by the National Science Foundation under grant numbers NCR-9105372 and NCR-9204853, the Advanced Research Projects Agency under contract number DABT63-93-C-0052, and an equipment grant from Sun Microsystems' Collaborative Research Program. The information contained in this paper does not necessarily reflect the position or the policy of the U.S. Government or other sponsors of this research. No official endorsement should be inferred.

9. Bibliography

[Andreessen 1993]
M. Andreessen. NCSA Mosaic Technical Summary. Tech. Rep., National Center for Supercomputing Applications, May 1993.

[Berners-Lee et al. 1992]
T. Berners-Lee, R. Cailliau, J. Groff and B. Pollermann. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 2(1), pp. 52-58, Meckler Publications, Westport, CT, Spr. 1992.

[Blair & Maron 1985]
D. C. Blair and M. E. Maron. An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System. *Commun. ACM*, 28(3), pp. 289-299, Mar. 1985.

[Borenstein & Freed 1993]
N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. Req. For Com. 1521, Sep. 1993.

[Bowman et al. 1993]
C. M. Bowman, P. B. Danzig, U. Manber and M. F. Schwartz. *Scalable Internet Resource Discovery: Research Problems and Approaches.* Dept. Comput. Sci., Univ. Colorado, Boulder, Oct. 1993. To appear, Commun. ACM. Tech. Rep. CU-CS-679-93.

[Bowman et al. 1994]
C. M. Bowman, C. Dharap, M. Baruah, B. Camargo and S. Potti. A File System for Information Management. Tech. Rep. CSE-94-023, Comput. Sci. Dept., Pennsylvania State Univ., Mar. 1994.

[Cate 1992] V. Cate. Alex - A Global Filesystem. *Proc. Usenix File Systems Workshop*, pp. 1-11, Ann Arbor, MI, May 1992.

[Clark & Tennenhouse 1990]
D. D. Clark and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. *Proc. SIGCOMM Symp.*, pp. 200-208, Philadelphia, PA, Sep. 1990.

[Danzig, Hall & Schwartz 1993]
P. B. Danzig, R. S. Hall and M. F. Schwartz. A Case for Caching File Objects Inside Internetworks. *Proc. SIGCOMM Symp.*, pp. 239-248, San Francisco, CA, Sep. 1993.

[Davis 1994]
J. R. Davis. *Dienst: A Distributed Interactive Extensible Network Server for Techreports.* Design Research Institute, Cornell Univ., Jan. 1994.

[Emtage & Deutsch 1992]
A. Emtage and P. Deutsch. Archie - An Electronic Directory Service for the Internet. *Proc. USENIX Wint. Conf.*, pp. 93-110, Jan. 1992.

[Foster 1992]
S. Foster. About the Veronica Service. Electronic bulletin board posting on the comp.infosystems.gopher newsgroup, November, 1992.

[Gifford et al. 1991]
D. K. Gifford, P. Jouvelot, M. A. Sheldon and J. W. O'Toole, Jr. Semantic File Systems. *Proc. 13th ACM Symp. Operating Syst. Prin.*, pp. 16-25, Oct. 1991.

[Hardy & Schwartz 1993]
D. Hardy and M. F. Schwartz. Essence: A Resource Discovery System Based on Semantic File Indexing. *Proc. USENIX Wint. Conf.*, pp. 361-374, Jan. 1993.

[ISO 1987] ISO. Information Processing Systems - Text Communication - Remote Operations - Part 2: Protocol Specification. Draft International Standard ISO/DIS 9072-2, International Organization for Standardization, 1987.

[Kahle & Medlar 1991]
B. Kahle and A. Medlar. An Information System for Corporate Users: Wide Area Information Servers. *ConneXions - The Interoperability Report*, 5(11), pp. 2-9, Interop, Inc., Nov. 1991.

[Kent 1993] S. T. Kent. Internet Privacy Enhanced Mail. *Commun. ACM*, 36(8), pp. 48-60, Aug. 1993.

[Manber & Wu 1994]
U. Manber and S. Wu. GLIMPSE: A Tool to Search Through Entire File Systems. *Proc. USENIX Wint. Conf.*, pp. 23-32, Jan. 1994.

[McCahill 1992]
M. McCahill. The Internet Gopher: A Distributed Server Information System. *ConneXions - The Interoperability Report*, 6(7), pp. 10-14, Interop, Inc., July 1992.

[Microsoft Inc. 1993]
Microsoft Inc. *OLE 2.01 Design Specification.* Microsoft OLE2 Design Team, Sep. 1993. Describes Object Linking & Embedding environment.

[Muntz & Honeyman 1992]
D. Muntz and P. Honeyman. Multi-Level Caching in Distributed File Systems — or — Your Cache Ain't Nuthin' But Trash. *Proc. USENIX Winter Conf.*, pp. 305-313, San Francisco, CA, Jan. 1992.

[NBS 1977] NBS. Data Encryption Standard. *Federal Information Processing Standards Publication 46*, National Bureau of Standards, U.S. Dept. of Commerce, Washington, D.C., Jan. 1977.

[Neuman 1992]
B. C. Neuman. Prospero: A Tool for Organizing Internet Resources. *Electronic Networking: Research, Applications, and Policy*, 2(1), pp. 30-37, Meckler Publications, Westport, CT, Spr. 1992.

[Ousterhout et al. 1985]
J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer and J. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. *Proc. 10th ACM Symp. Operating Syst. Prin.*, pp. 15-24, Dec. 1985.

[Postel & Reynolds 1985]
J. Postel and J. Reynolds. File Transfer Protocol (FTP). Req. For Com. 959, USC Information Sci. Institute, Oct. 1985.

[Salton 1986]
G. Salton. Another Look at Automatic Text-Retrieval Systems. *Commun. ACM*, 29(7), pp. 648-656, July 1986.

[Sun 1987] Sun. XDR: External Data Representation Standard. Req. For Com. 1014, Sun Microsystems, Inc., June 1987.

[Tanenbaum 1988]
A. S. Tanenbaum. *Computer Networks.* Prentice Hall, Englewood Cliffs, NJ, 1988. Second edition.

[USENIX Association 1986]
USENIX Association. UNIX Supplementary Documents. 4.3 Berkeley Software Distribution, Nov. 1986.

[Weider, Fullton & Spero 1992]
C. Weider, J. Fullton and S. Spero. Architecture of the Whois++ Index Service. Internet Draft, WNILS Working Group, Nov. 1992. Available by anonymous FTP from nri.reston.va.us, in internet-drafts/draft-ietf-wnils-whois-00.txt.

[Welch 1984]
T. A. Welch. A Technique for High Performance Data Compression. *IEEE Computer*, 17(6), pp. 8-19, June 1984.