Analyzing Image Content for a Large Scale Hypermedia System

John H. Boose, Larry S. **Baum,** Randy J. **Kelley**

The Boeing Company, **7L-44,** PO Box **24346,** Seattle, WA **98124**
john@redwood.rt.cs.boeing.com, lbaum@redwood.rt.cs.boeing.com, rkelley@redwood.rt.cs.boeing.com
Boeing's Intelligent Graphics team also includes Mike Carter, Susan Chew, Chuck Gebhardt, Brent Hadley, Ed Hall, Tom Jenkins, Cathy **Kitto,** Steve Pruitt, and Dave **Shema**

Abstract

The Boeing Company maintains tens of millions of pages of information associated with the manufacture and delivery of its products. Much of this information must be made available electronically. We have developed tools to automatically convert and integrate electronic data into industry standard formats. Some of the technical challenges include 1) handling a wide variety of source formats, 2) making sure that the tools scale up to handle millions of pages of information, and 3) adding functionality to graphics. Our system contains ***over*** *four million pages of text including tens of thousands of graphics.*

In this paper we describe tools that recognize and use information within airplane-related vector and raster images. Such images include troubleshooting charts, fault reporting diagrams, component location diagrams, component index tables, wiring diagrams, system schematics, parts illustrations, standards tables, and structural and tooling drawings. Each airplane requires conversion of over 20,000 graphics including over 900,000 pieces of cross-referenced information. We are also exploring visual information retrieval strategies, including content-based and similarity-based methods for both vector and raster graphics.

Keywords: information retrieval, hypermedia

1 Four Million Pages and Growing

The Boeing Company is committed to Air Transport Association Specification 2100 (ATA, 1995) which requires the use of the *Standard Generalized Markup Language* (SGML) **(Goldfarb, 1990).** To comply with these specifications Boeing has converted millions of document pages to SGML. Document sources include commercial tools (e.g., Microsoft Word, InterLeaf, FrameMaker), internal proprietary tools, and data from external vendors.

Manually converting such documents would be prohibitively expensive and error-prone. To meet this challenge our team built a series of text *autotaggers* that automatically convert documents into SGML. From **1990** to the present we developed autotaggers to create a testbed hypermedia system containing data from over four million pages. We use Electronic Book Technologies' DynaText system as the browser, including custom extensions for viewing graphics. We will report our text autotagging methods and results elsewhere.

2 Graphic Navigation and Functionality

The autotaggers receive text in various formats. They receive graphics in vector formats (usually CGM) and raster formats (usually TIFF) and originally did little more than pass the graphics to viewers in DynaText. The graphics lacked information for navigation and retrieval. Users often needed to examine many images to find the right one, including multiple panning and zooming operations. Hyperlinking from a text reference in a graphic was not possible, nor was full text search. Graphics lack the basic functionality of the SGML text. The graphics **also** lacked useful functional information ~ What is the logic in this troubleshooting diagram? What are the relationships between items in this table? What lines in this wiring diagram represent continuous circuits?

We built a series of graphic recognition tools that use vector and raster graphics to produce navigational and functional information represented in SGML. We then integrated this information into the testbed.

3 Graphics Recognition Tools

We used a similar approach for each graphic recognition problem:

- Develop a tool to reason about vector or raster graphics.
- Tune and evaluate the **tool** for accuracy and the ability to scale up.

Proceedings of the Second Australian Document Computing Symposium, **Melbourne,** Australia, April **5, 1997.**

- Integrate acquired graphic information with text in the testbed.
- Evaluate the usability of the results.
- Transfer the results to products and services.

The image counts given below are for a recent Boeing 747 airplane maintenance manual set. They illustrate the large scale of the problems.

Troubleshooting charts are vector drawings that provide mechanics with decision trees for fault isolation. One drawing can consist of as many as 30 individual sheets with intersheet connections linked by references. Our software analyzes the layout of the boxes and the arrows connecting them and builds an internal representation of the decision tree. It then generates hot spots so that users can easily navigate within a sheet and among sheets as well as follow links to other information. This manual set contains over 750 charts containing over 16,000 hot spots.

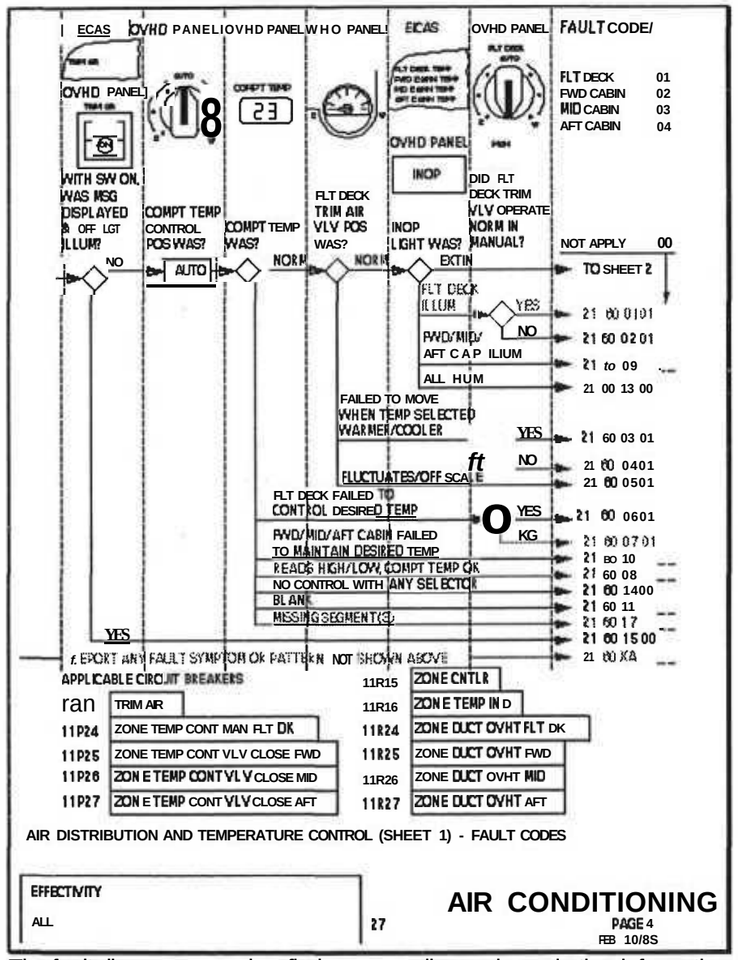


Figure 1. The fault diagram recognizer finds arrows, diamonds, and other information, then uses knowledge about diagram layout to produce the application's flow of logic.

Fault reporting diagrams are vector images that pilots use to determine what messages to record in the flight logbook when a malfunction occurs. The variations in layout are more complex than the troubleshooting charts; still we've been able to accurately determine the decision networks in these diagrams and generate over 20,000 hot spots in over 1,600 diagrams (Figure **1).**

Component location diagrams provide exploded views of aircraft and their components. Typically one diagram consists of multiple sheets, each of which has one or more subpictures (insets) with internal callout references. Our software must reliably subdivide each image into subpictures and generate the hot spots that link the callouts to the correct subpicture, including references to other sheets. In addition, the software generates hot spots around each equipment number, so that mechanics can easily navigate from the picture to information about that piece of equipment. This manual set contains over 650 vector component location diagrams; the system produced over 3200 subpictures containing over 25,000 hot spots.

Component index tables are vector drawings that tell the mechanic where to go to find the maintenance procedures for the equipment illustrated in the component location drawings. The index table recognizer determines the individual table cells and relationships among cell contents. It generates over 14,000 hot spots in over 300 tables linking the drawings to component location diagrams, other component index tables and to maintenance procedures (Figure 2).

Wiring diagrams are vector drawings extracted from computer-aided design data sets. The intelligent graphics software analyzes both the vector image and the data set to determine how wires are laid out in the drawings so that circuit tracing is enabled in the hypermedia viewer. In addition, there are hot spots providing hyperlinks to equipment lists and wire lists as well as other diagrams. This manual set contains over 1400 diagrams with over 135,000 hot spots (Figures **3, 4).**

System Schematics are similar in functionality and recognition requirements to wiring diagrams. This manual set contains over 1300 system schematics with over 461,000 hot spots.

Parts illustrations are *raster* images that provide exploded views of the airplane much like component location diagrams. They contain callouts and item numbers that must be linked to other pages. Because they are raster images we must perform more complex analysis than with the vector images. We have developed routines to quickly process callouts and item numbers with high accuracy. This manual set contains over 13,000 sheets with more than 241,000 hot spots (Figure 5).

Standards tables contain information about Boeing engineering practices. The sources for many of these tables are paper copies that were several generations old, made with typewriters and ruled with ball point pens. The conversion team scanned these images for on-line use and was planning to manually re-enter the information. We developed recognition software to extract the logic in the tables. Together with a commercial optical character recognition package we automatically converted over 6600 tables to InterLeaf format (Figures 6, 7). Later these will also be converted to SGML.

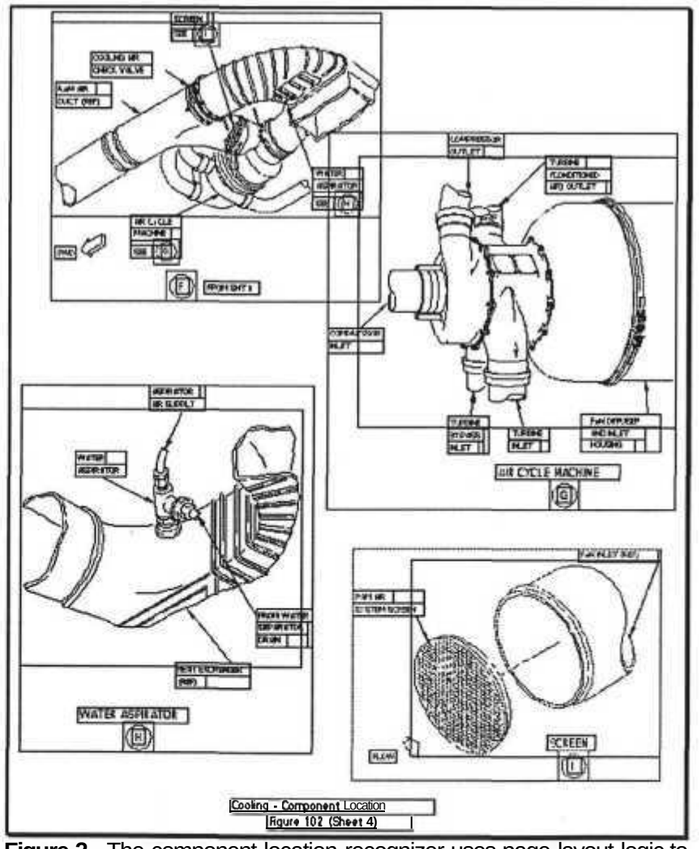


Figure 2. The component location recognizer uses page layout logic to induce groups and attach callouts.

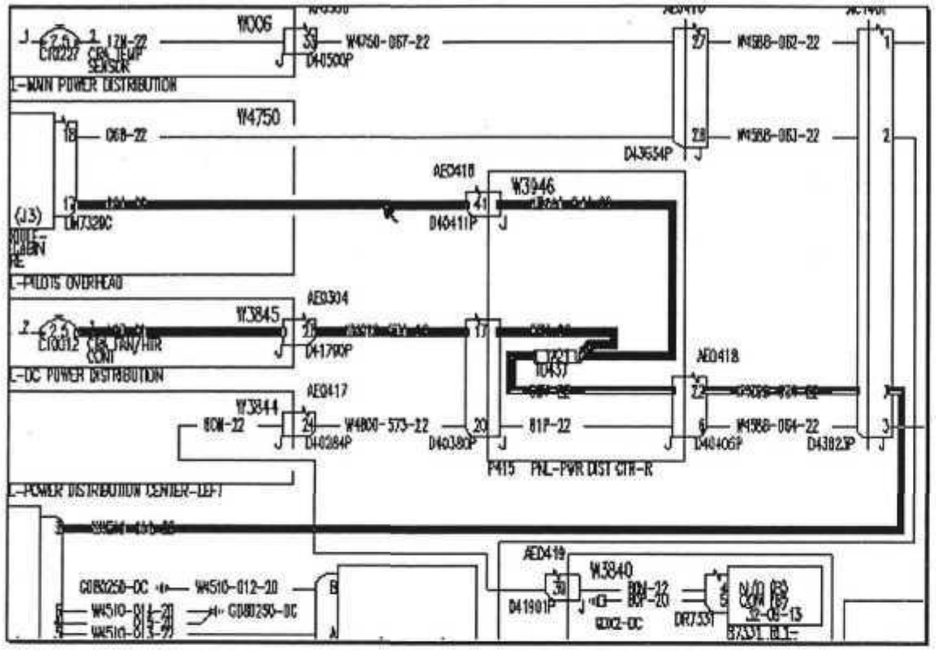


Figure 3. A wiring diagram recognizer finds connected circuits and links references together.

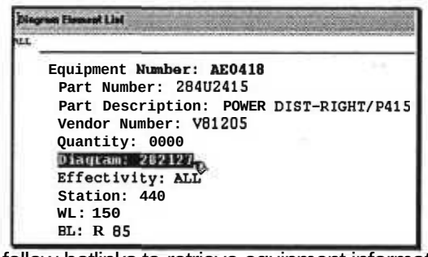


Figure 4. Users follow hotlinks to retrieve equipment information. The system stores wiring diagram links and other references in the DynaText database.

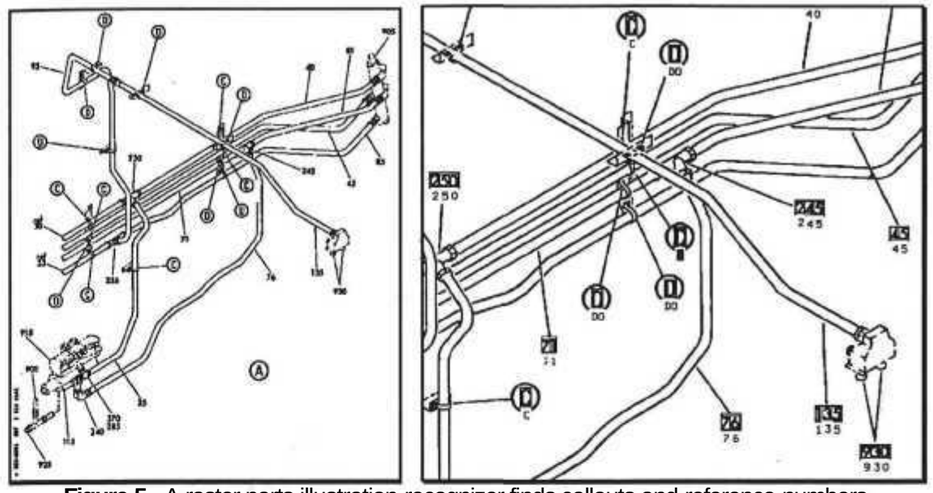


Figure 5. A raster parts illustration recognizer finds callouts and reference numbers, using symbol finding and optical character recognition.

Filler Metal Classifications			Procurement Specifications (information only)	Color Code 1		
AWS Class	Government Class	Trade Name or Type		Group Color	Primary Color	Secondary Color
ERAZ02A	AZ02A	AZ02A	AWS A5.12 2	Yellow	Orange	NOB*
ERAZ61A	AZ61A	AZ61A		Yellow	Grey	None
ERFZ33A	FZ33A	FZ33A		Yellow	Pink	None
ERAZ101A	AZ101A	AZ101A		Yellow	Blue	None
-	LA141A	LA141A	AMS 4897 2	Yellow	Black	None

Figure 6. This original standards table was made using a typewriter and ball-point pen.

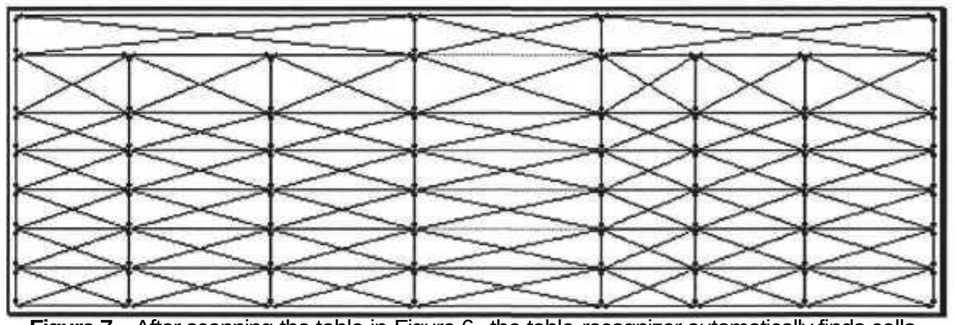


Figure 7. After scanning the table in Figure 6, the table recognizer automatically finds cells, performs character recognition, then produces standard mark-up files.

4 Evaluating Recognition Results

To evaluate the results of text tagging our team used commercial SGML verifiers and specialized custom systems. We developed tools that analyze the text output for consistency and completeness, including statistical measures and link verification. Often these tools find authoring errors as well as tagging errors.

It is more difficult to build such tools to check graphic recognition accuracy. We built a custom graphic analysis tool that simultaneously shows recognized objects and the corresponding SGML (Figure 8). For example, if the recognizer finds a reference, the viewer displays the hot spot along with the information as to what type of reference it is and to what it refers. Using this viewer we manually sampled thousands of images to assess recognition accuracy. Recognition accuracy ranged from about 97% for component location graphics to 99.9% for troubleshooting charts. These results exceeded our performance goals.

We can also find some types of graphics authoring errors. For example we can automatically detect incorrect references that point to non-existent locations. Previously, such errors could only be found by manual inspection.

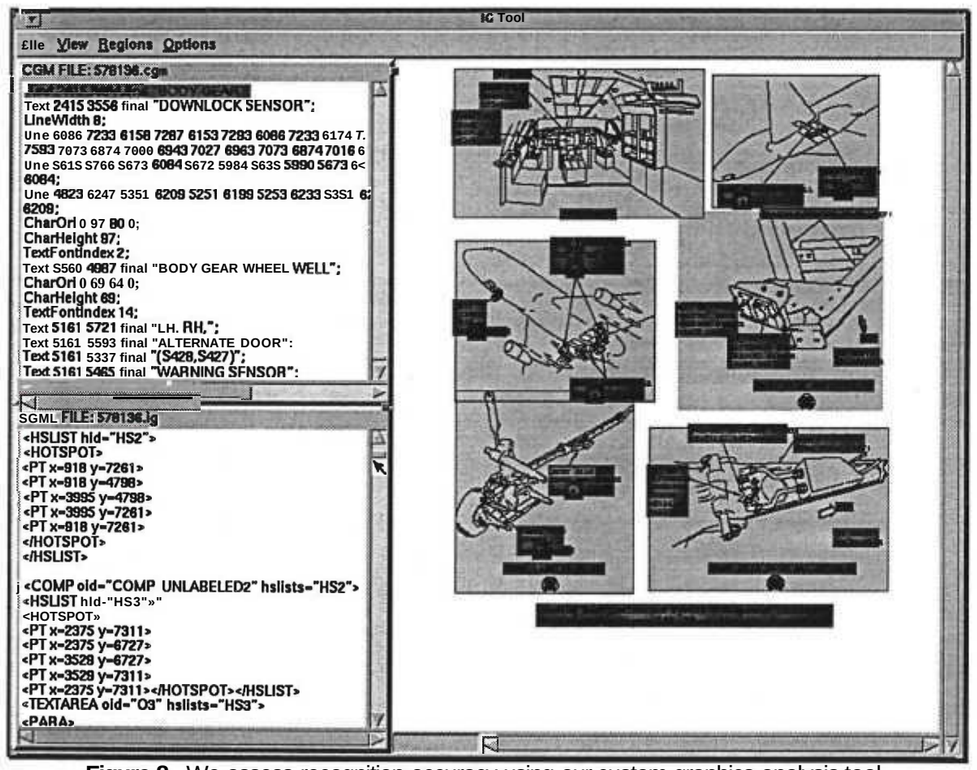


Figure 8. We assess recognition accuracy using our custom graphics analysis tool.

5 Future Challenges - Work in Progress

We are currently working with Boeing's library of millions of structural and tooling raster drawings. These drawings have been scanned from aperture cards and placed on-line. We would like to recognize objects such as title blocks, part numbers, dimension lines, materials lists, and parts tables. Much of the text is hand written and many of the scanned images are skewed or noisy. This is challenging work (Figure 9).

We are also working on visual information retrieval strategies, including content-based and similarity-based methods (Figure 10).

6 Summary

We have successfully added functionality to tens of thousands of vector and raster graphics, integrating them with text in on-line navigation systems, addressing the problems of scale and accuracy. We hope to continue to add functionality to graphics to enhance the overall utility of Boeing's digital data.

References

ATA (1995). Air Transport Association Specification 2100 - Digital Data Standards for Aircraft Support, 1995, Order code A090

Goldfarb, Charles F. (1990). *The SGML Handbook*, Oxford University Press, ISBN 0-19-853737-9

Figure 9. The structural and tooling drawing recognizer must find part numbers, title blocks, and dimensions lines in raster images.

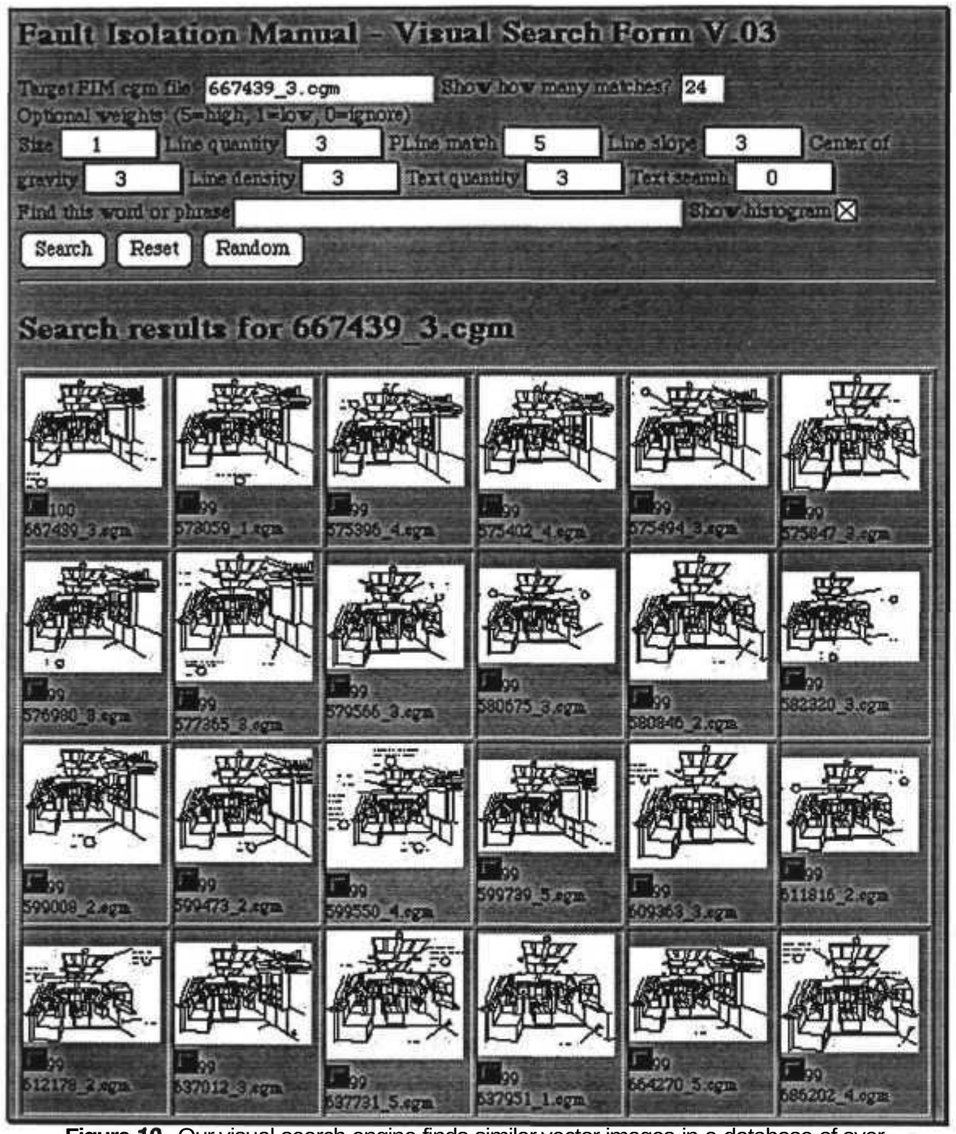


Figure 10. Our visual search engine finds similar vector images in a database of over 6,000 drawings for oneFault Isolation Manual.

A Multi-party Document Model

Seng-cho T. Chou

Department of Information Management
National Taiwan University
Taipei, Taiwan, R.O.C.
chou@ccms.ntu.edu.tw

Abstract

A multi-party *document (MPD) is one that contains multiple parts intended for multiple recipients in that each recipient does not necessarily need to know the existence, and therefore the corresponding content, of the other parts of the document intended for other recipients. MPDs pose new requirements, namely, transparency and security, to the underlying system. This paper reviews the traditional document models and their shortcomings with respect to the stated requirements and proposes a new model that satisfies those requirements. The major implementation issues including document construction and encryption for MPD supports are also discussed. Shifting these capabilities to the system level will relieve burdens of users and enhance the security measure of the system for protecting the integrity and privacy of document contents. The issues discussed in this paper are important for making a sound underlying support system for the development of information systems, in particular in the Intranets environment.*

Keywords Document management, document structure, security, encryption, Intranets.

1 Introduction

Traditionally, a document such as an electronic mail, an HTML document on the Internet or the like is to be treated as one to be read by one or more people as one whole document. From the security standpoint, the traditional encryption systems such as the Public Key system [1, 2] or the Private Key system [3] that encrypts and decrypts a document at the sender and receiver ends respectively work well for these documents. We refer to these documents as *single-party* documents (SPD) to reflect the fact that the whole document is addressed to one single-party at a time, disregarding whether there are multiple recipients for the document or not. A *multi-party* document (MPD), in contrast, is one, which contains multiple parts intended for multiple recipients in that each recipient does not necessarily need to know the existence, and therefore the corresponding content, of the other parts of the document intended for other recipients.

Multi-party documents find many applications in our daily life. For example, a grade sheet is an MPD in that the grades contained in such a document for the whole class of students are the parts intended for each of the students in the class. Each student can read his/her own grades but not the others'. In other words, it is desirable that the privacy of a student's grades is observed regardless of how the grade sheet is to be received, either being delivered to the students or accessed by students on an on-demand fashion. Similarly, a classified project planning document might contain parts covering status and assignments for each team member, who does not need to know the existence of any other team members. The protection of this document from being misused by team members or outsiders has profound security implications. In particular in an environment where document-based workflow is common and essential such as the work group environment [4, 5] promoted by the current popular Web [6] and Internet, or Intranets [7], technology, the appropriate support for multi-party documents will have a tremendous impact on the success of the ultimate systems.

MPDs therefore pose new requirements to the underlying system. The two major requirements are: First, the security measure of an MPD should not be compromised; neither an MPD be accessed nor handled by an unnecessary third party. Secondly, for document manipulation purpose, the illusion that an MPD be a single document should be supported; artificial boundaries imposed on an MPD is not desirable.

This paper reviews the traditional document models and their shortcomings with respect to the requirements stated above and then proposes a new model that satisfies those requirements in Section 2 and 3 respectively. We will also highlight the implementation considerations of such an environment that supports MPDs in Section 4 before the conclusion is drawn in Section 5.

Proceedings of the Second Australian Document Computing Symposium, Melbourne, Australia, April 5, 1997.

2 Traditional document models

Abstractly, let $D = \{d_1, d_2, ..., d_n\}$ be a document consisting of document blocks d_i's. Let *R(d)* be the set of recipients of the document block *d.* If

$$R(d_i) = R(d_j), \forall i, j$$

then *D* is referred to as a single-party document and otherwise, a multi-party document. Note that both single-party and multi-party documents can have more than one recipient but the content of the former is intended for all recipients.

The requirements for multi-party document supports can be stated as: (1) Security. Each document block, d_i, of a document *D* be read by those recipients who are in *R(di)* and not anyone else. In other words, for any recipient r,

$$D_r = \{d_i | d_i \in D \wedge r \in R(d_i)\}$$

is the document to be seen by *r;* (2) Transparency. *D* be treated as a single document by the document owner or creator regardless of whether *D* is a single-party or multi-party document.

2.1 The SM/M model

A common approach to handle MPDs traditionally is based on the mail-merge or multiple documents model. A mail-merge document consists of a document template and a database. There are place-holders in the template for information from the database. Then, for each of the qualified instances retrieved from the database, the mail-merge program will put the attribute values into the appropriate place-holders and, in effect, creating a new copy of specialized document. Consequently, a collection of specialized documents is created.

What the mail-merge program does is to turn a single document into multiple documents. Each of these documents is then delivered to its recipient separately. We call this model a SM/M model where SM refers to the conversion of a single document into multiple documents by the sender and the latter M reflects that multiple documents are delivered.

With respect to the two requirements for MPD supports, the mail-merge model satisfies the security requirement in that each specialized document is delivered and read by one and only intended recipient; no third party is involved. However, the requirement that the multi-party document be treated as a single document is compromised. First, a mail-merge document involves a document template and a database, which are often two separate document entities; the document owner has to deal with these two entities with some efforts rather than just treating it as one single document. Moreover, after the conversion, a collection of documents is formed; the owner has to deal with this set of documents rather than one single document should the owner decide to work with individual documents. Not satisfying the transparency requirement puts the burden onto the user.

2.2 The S/SM model

Another type of representative MPDs often encountered is a grade sheet, which is often handled by a grade report model. A grade sheet can be viewed as a table where the information in each row is meant for a particular person and therefore the privacy of information is meant to be kept. So, sending the grade sheet to everyone on it is not acceptable. Instead, a grade sheet is often submitted as a whole to some authority such as the registrar office where grades on the sheet are extracted and recorded onto the respective students' grade reports, which are then sent to the individual students for reviewing.

A grade report model is referred to as the S/SM model to reflect the fact that a single document is created and manipulated by the owner. It is also shipped out as a single document by the sender. The document is then converted or incorporated into multiple documents by an appropriate computer program or manual process.

The grade sheet allows the owner to view it and manipulate it as a single document although it is an MPD meant for multiple recipients. This satisfies the transparency requirement for MPD supports but the security requirement is compromised. Since a third party (outside of the system) is involved in the conversion of a single document into multiple documents, the privacy of the information in the original grade sheet cannot be guaranteed.

3 The S/SM/M model

The shortcomings of both the mail-merge and grade report models discussed above suggest the need for a new model. We propose an S/SM/M model in this section.

Combining the spirit of the S/SM and SM/M models, the S/SM/M model emphasizes that an MPD should be manipulated as a single document by the document owner (the leading S) and the recipient is to receive a single-party document and therefore multiple documents are to be generated by the system (the trailing M). In addition, the conversion of the source MPD to multiple single-party documents is the sole responsibility of the system with no involvement of any other third party program (the middle SM). The proposed model will therefore satisfy both the security and transparency requirements stated above.

A simple example document structure that follows this model would be an MPD consisting of tagged blocks where a tag indicates who the recipients are for the corresponding block. This basic MPD can assume a simple flat document structure. The document owner will simply create tags along with blocks; the whole MPD is one document as far as the owner is concerned.

Bearing the same tagged-block structure, an HTML document structure can easily be extended to support the S/SM/M model by adding an MPD tag to it. This has an important implication as HTML documents are becoming more and more popular under the current network computing trend. Here we give two examples to show the use of multi-party HTML documents. The first example deals with the administration of homepages on the Web. Currently, the homepages of most Web sites are fixed; consequently, for any of such Web sites, the same homepage will be given to everyone who connects to it. In some Web sites, their system administrators might want to tailor their homepages according to their clients' needs or preferences, ranging from some small items or wordings to the whole page, or from the language to the style of presentation. The current practice is based on a programming approach; the system administrator is responsible for providing or employing some appropriate mechanism, such as Java applets or CGI programs, suited to his/her needs. In contrast, with the MPD support, the system administrator can produce the needed homepage easily and naturally.

A second example has to do with the HTML document-based (paperless) Intranets document workflow environment. In such an environment, a document might be distributed to or travel through a handful of recipients who might assume different access rights to the different parts of the document. An MPD would be a perfect solution here as opposed to the need for the document sender to create multiple documents for multiple clients. In fact, it might not even work sometimes to replace a workflow document with multiple documents. An MPD allows the document owner to write it as one document and let the system take care of the protection of the different parts of the document for individual access.

We should emphasize again that it is the responsibility of the system to make the conversion of an MPD into multiple SPDs happen. There could be many different kinds of MPD document structures. Systems must bear the appropriate conversion capabilities in order to honor the transparency requirement for MPD supports.

4 Implementation issues

In this section, we discuss the major issues that need to be considered in order to implement MPD supports. The two major aspects are document construction and encryption. We first distinguish two modes of access below.

4.1 Mode of access

There are two common document access modes, namely, the on-demand retrieval and the distribution mode. The former favors documents to be retrieved by their recipients based on an on-demand fashion. For example, students might check their grades on their own desire, or in the Web environment, HTML documents are retrieved on clients' discretion. In contrast, the latter supports the activity of distributing documents to all recipients once the documents are done. The Intranets document workflow system is a typical example of this kind. The document access modes can have an effect on the document construction and encryption approaches.

4.2 Document construction

The nature of the on-demand retrieval access implies that an MPD is not necessarily accessed by all recipients at the same time or might not be seen by some recipients at all. This would mean that converting an MPD into multiple single-party documents for each of the recipients might not be necessary or economical. In fact, it might be desirable to allow the parts of an MPD for any particular recipient to be generated dynamically. On the contrary, the nature of distribution access mode suggests that it is necessary to distribute the MPD to all recipients. Therefore, it is a good idea to convert an MPD into multiple single-party documents for distribution.

The mode of access and the document structure are not necessarily independent. A flat-structure MPD is one consisting of non-overlapping tagged blocks. A flat-structure MPD would need to be scanned once for each on-demand access request. In case of distribution access, it is necessary to scan through an MPD more than once in order to sort out the blocks for multiple recipients due to the fact that a block can have more than one recipient.

Other document structures are possible. For example, a grade-sheet MPD can assume a document structure similar to a mail-merge document, consisting of a grade-sheet table and some command that specifies how each of the rows in the table can be accessed by the corresponding recipient. Then, for distribution access, documents can be generated and distributed. But for on-demand access, a look-up feature might be helpful in locating the information for the dynamic generation of the document for that particular recipient.

Table 1 summarizes the relationship between document access modes and document construction considerations. As it indicates, the on-demand access favors dynamic document construction along

with some look-up capability to speed up the information extraction process. Whereas, the distribution mode of access favors full conversion; therefore, it will be beneficial to have efficient algorithms to help out the document conversion and construction process.

	On-demand	Distribution
Access nature	dynamic construction	full conv.
Flat-structure	scan once	scan more than once
Grade-sheet structure	dynamic construction with look up support	full conv.

Table 1: The relationship between access modes and document construction.

But in any case, it is the responsibility of the system to support a friendly environment for the users such that the users will have the illusion of working on a single document regardless of the nature of access, the document structure, and document type, either SPD or MPD.

4.3 Encryption

To protect the integrity and privacy of document contents, encryption is commonly used. The two common encryption systems are the public key and the private key systems along with the popular RSA and DES encryption algorithms respectively. In a public key system, the document server encrypts a document using the public key of a particular client (document recipient) and then the recipient decrypts the encrypted text with its own private key. The client's public key is made available to the public by registering it in some public certificate authority. The public key system is also known as the two-key system or asymmetric-key system. In the private key system, the server and the client would need to agree on the secret encryption key to be used in advance. Then, a document will be encrypted and decrypted using the same secret key. The private key system is also called the one-key system or the symmetric-key system. The public key system is considered more dynamic since keys can be changed as desired and there is no key distribution problem that the private key system would have to face.

In what follows, we discuss the protocols between the server and clients to get MPDs from the server to clients under the two different encryption systems and access modes.

- Public key system with on-demand access
  - A client C, with a pair of public and private keys P_u and P_r respectively, sends a request to the server S for an MPD M;
  - S generates an SPD D from M, encrypts D with C's public key P_u to get D', sends the encrypted document D' to C;
  - C decrypts D' with its private key P_r to retrieve the original document D.
- Secret key with on-demand
  - A client C and the server S agree on the secret encryption key R to be used in advance.
  - C sends a request to S for an MPD M;
  - S generates an SPD D from M, encrypts D with the secret key R to get D', sends the encrypted document D' to C;
  - C decrypts D' with the same secret key R to retrieve the original document D.
- Public key with distribution
  - Each client (recipient) has its own pair of public and private keys (see below for a discussion of group concept).
  - The server S generates a set of single-party documents SD from some MPD to be sent to each of the intended recipients. For each document $D_r \in SD$, S encrypts D_r with the public key that belongs to the recipient r of D_r to get D'_r, sends the encrypted document D'_r to r;
  - for some particular client C who receives an encrypted document D'_c, the client decrypts D'_c with its private key to retrieve the original document D_c.
- Secret key with distribution
  - Each client and the server S agree on some specific secret encryption key to be used in advance (see below for a discussion of group concept).
  - S generates a set of single-party documents SD from some MPD to be sent to each of the intended recipients. For each document $D_r \in SD$, S encrypts it with the secret key that belongs to the recipient r of D_r to get D'_r, sends the encrypted document D'_r to r;
  - for some particular client C who receives an encrypted document D'_c, the client decrypts D'_c with its secret key to retrieve the original document D_c.

From the above discussion, we can see that both the public key and private key encryption systems work well to support MPDs.

4.4 Single encryption key and group concept

Normally, each client has its own key for the encryption purpose such as the secret key in the private key system or the pair of public and private keys in the public key system. When a server needs to work with more than one client, the server would need to deal with a collection of encryption keys, one for each of the recipients. This can become a burden to the server. Fortunately, this situation can be relieved with some simple mechanism. A group concept is seen in many different situations in the computing world. This same concept can be used here for grouping the recipients of documents under certain situation. Then, the key for all recipients in some particular group can be the same, allowing the server to use just one key. This applies to both the public key and private key systems. In other words, the server can use the same public / secret key to encrypt documents intended for all the members of some particular group in the public / private key system respectively. This discussion is mainly for the distribution access mode only. It should make practically no difference in performance in the case of on-demand access since only one recipient is involved in this case.

There is a slight difference in the use of a single encryption key for a group of recipients as opposed to specific keys for individual recipients. In the former situation, any encrypted document can be decrypted by any recipient in the group since all recipients possess the same private or secret key. So, in this case, encryption protects documents from being accessed by outsiders only. Whereas in the latter situation, an encrypted document for a particular recipient can only be decrypted by that recipient but not anyone else, not even other members in the same group. So, the latter exercises a stricter security measure than the former.

5 Concluding remarks

The main emphasis of this paper is to identify the limitations of current systems in handling multi-party documents, and to stress that shifting the MPD handling capabilities to the system level will relieve burdens of users and enhance the security measure of the system for protecting the integrity and privacy of document contents. The issues discussed in this paper are important for making a sound underlying support system for the development of information systems, in particular in the Intranets environment. As network computing and Intranets applications promise to become more and more important, and the document flow is an important part of such a computing environment, the appropriate MPD supports can significantly contribute to the ultimate success of these systems.

Our future work includes the development of processing modules for multi-party HTML documents, and the incorporation of the proposed S/SM/M model into our Intranets document workflow system being developed, in particular where the protection of document contents is a major concern. To enrich the document structure is also our another focus. As an example, a document could also be considered to be composed of document block objects. Tags would be a special attribute of block objects. To create documents means to create block objects and to fill in the tag attributes, among other things. This object-based document structure is much richer in comparison to the flat one mentioned earlier, and can be the basis for supplementing a document editor.

References

[1] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, Vol. IT-22 No. 6, Nov. 1976.

[2] A. E. Hutt, S. Bosworth, and D. B. Hoyt. *Computer Security Handbook*, John Wiley & Sons Inc., third edition, pages 16.1-16.17, 1995.

[3] Data Encryption Standard (DES). *Federal Information Processing Standards Publication* 46-2, National Institue of Standards and Technology (NIST), December 1993.

[4] C.A. Eills, S.j. Glbbs, and G.L. Rein. Groupware: Some Issues and Experiences. *CACM*, Vol. 34, No. 1, pages 38-58, 1991.

[5] J. Grudin. CSCW Introduction. *CACM*, Vol.34, No.12, pages 30-34, 1991.

[6] T. Berners-Lee, et al. The World-Wide Web. *CACM*, Vol.37, No.8, pages 76-82, 1994.

[7] Intranets Redefine Corporate Information Systems. Netscape white paper, http://home.netscape.com/comprod/at_work/white_paper/indepth.html, 1996.

LivePAGE - A multimedia database system to support World-Wide Web development

Donald D. Cowan
Department of Computer Science
University of Waterloo
Waterloo ON N2L 3G1
E-mail dcowan@csg.uwaterloo.ca

Daniel M. German
Department of Computer Science
University of Waterloo
Waterloo ON N2L 3G1
E-mail dmg@csg.uwaterloo.ca

Eric Mackie
Inforium Technologies Inc.
158 University Ave. W.
Waterloo ON N2L 3E9
E-mail eric@inforium.com

Abstract

The rampant growth of the World-Wide Web (WWW) is largely a consequence of its simplicity. A typical person can quickly learn HTML and start creating WWW pages in an afternoon. As WWW sites become larger and more complex, this inherent simplicity causes multiple problems as many of the current tools and techniques are stretched to address issues they were not designed to handle. These problems are compounded by the rapid proliferation of solutions which are often fairly ad-hoc in nature. In this paper we present a layered model which through its tools and techniques provide a more disciplined approach to constructing and maintaining a WWW site. We then describe an implementation of this model which is based on two more mature technologies; SGML and relational database systems.

Keywords Architecture, World-Wide Web, multimedia, SGML, Web site development, document database, hypermedia.

1 Introduction

The accessibility of simple-to-use, but powerful interfaces or browsers, and the apparent simplicity of HTML has prompted the development of literally millions of hypermedia World-Wide Web (WWW) application sites. In many of these applications both the size and number of the source documents are sufficiently small that each application can be developed and maintained by a single person without the assistance of any methodology or tools (except for a text editor). In contrast, the developer of a large WWW site is constantly struggling to master the complexity involved in the design, development and maintenance of such a site with scores of pages and links.

The WWW is a large collection of interrelated resources, linked through WWW pages tagged using HTML (HyperText Markup Language [3]). HTML pages can point to other resources on the WWW, such as images, video, sound and text. Each resource on the WWW has, at least, one address, known as a URL (Uniform Resource Locator, see [4]). URLs are strongly tied to the file system of the machine in which they reside, and often rely on common differences between file systems such as case-sensitivity and the length of file names. This reliance makes portability of documents between machines difficult.

The WWW is quite relaxed about the type of its components. Although HTML is defined by an SGML DTD,[1] an HTML document rarely complies with this definition.[2] Consequently, a WWW server normally relies solely on the extension to a file's name to determine its type.

Because of the focus on the markup language HTML, most of the tools produced to date are oriented toward editing HTML files. Very little research has been directed toward authoring systems that manipulate WWW sites as a collection of nodes and links, and that view HTML more as a presentation language than as a storage format.

The WWW is thriving mainly because of its simplicity. A typical person can quickly learn HTML and start creating WWW pages in an afternoon. HTML was meant to be a generic structural language; but power users felt it was too restrictive, especially for presentation purposes. Since its appearance, HTML has been expanded to support more and more presentation features including executable code (Java and JavaScript) and style-sheet information (Cascading Style Sheets -CSS [5]-). These new features while enriching the capabilities of the WWW are making WWW documents more and more difficult to read, write, and maintain.

The W3 Consortium (the body in charge of standardizing and promoting the WWW[3]) has always claimed that simplicity is not an objective for any new WWW standards, because tools can isolate the user from such complexities. For instance, HTML files with "frames" and "style-sheets" might be difficult to read, but a tool such as a WYSIWYG editor can be used to create and display them in a more readable format.

Most tools to support authoring of WWW applications have been focusing on the HTML file as the main entity. The authors of these tools pay almost no attention to the fact that HTML pages are just a visual representation of more abstract entities and that, in a WWW site, entities are interrelated, and the modification of one entity might require changes in another. The following list highlights some of the common problems that are a consequence of using the file as the only medium for the WWW hypermedia system:

- Consistency. Since every page is independent, information common to several pages has to be repeated, presenting a potential consistency problem.[4]
- Organization. The organization of a WWW site is bound at the time that the files are created. Restructuring is expensive and requires not only splitting, collating, renaming or deleting files, but updating links among files.
- Navigation. Since the navigational structure is embedded in the files, it is difficult to find and modify.
- Presentation. Information is not isolated from its presentation.[5] If the presentation of a set of pages needs to change, that information has to be changed on a file-by-file basis.
- Referential Integrity. When the URL of a resource changes, all the links pointing to it have to be updated. This problem is common both, at the global level, where the author does not have control over the documents pointing to the local information from an external site, and at the local level.

The apparent simplicity of HTML has a substantial cost. The author mixes together content, presentation and navigation information indiscriminately, and then stores the result in single file. Because of this integration, it is difficult to change a WWW site's presentation, structure, and content without a complete rewrite. In addition, because there is no clear distinction among these disparate concepts, a WWW site designer often considers the navigational structure first, followed by presentation, and finally, if there is any time left, the content. Ideally, this process should be reversed: write the content, define the presentation, and then impose a navigational structure.

2 WWW development systems

Different approaches have been tried to separate the content from its organization and presentation. We highlight some of them in this section.

2.1 Other tagging languages

Since HTML has limited structural components and is more oriented toward presentation, some authors have chosen to use another tagging language for the "master" of the information they maintain. The most common tagging system chosen uses SGML-compliant tagging languages which have a richer structure than HTML. These languages allow the author to characterize the structure of a set of documents, and to enforce this structure. Using ad-hoc filters, the SGML files can be converted to HTML. The advantage of such an approach is that the structure of the published information is kept separate from its presentation. If the user decides to change the master document's appearance, then only the filters need to be changed. A number of companies who publish the same information in various forms such as HTML, paper, PostScript or Acrobat, have chosen SGML-compliant tagging languages as their master format.[6] Other master formats have been proposed: LaTeX, word-processor based tags, and RTF. All these consider a WWW site as a text document, with well specified rules to translate it into HTML. Properly used, this

[1] See [10] for a description of SGML and DTDs

[2] Bray found that only 4.9% percent of 3 million HTML documents he analyzed have a <!DOCTYPE declaration (which is required in a SGML document). It is unknown what percentage of these were actually compliant with any of the HTML DTDs [6].

[3] http://www.w3.org

[4] Some HTTP servers support "server-side includes", which are directives to the server that are replaced by the contents of a given file or the result of an executable program; this only partially addresses the problem of consistency at a high performance price, since the directive is executed every time the node is requested.

[5] Cascading Style Sheets are trying to solve this problem partially at the file level.

[6] At the Developer's Workshop "Using SGML on the World Wide Web", during the 5th International WWW Conference, Sun Microsystems and Novell Inc. acknowledged that they have taken this approach to deal with the complexity of publishing their manuals.

Proceedings of the Second Australian Document Computing Symposium, Melbourne, Australia, April 5, 1997.

approach avoids inconsistencies, and separates presentation and navigation information from content. The main disadvantage of this approach is its focus on documents with a linear structure, such as books and articles, rather than the highly interconnected structures of objects typical in hypermedia applications.

2.2 Publishing SGML-compliant documents directly on the WWW

Some WWW publishers make their documents only available for users with specific SGML browsers (such as Grif Symposia [16] and SoftQuad's Panorama). They publish documents tagged with SGML-based tagging languages compliant with a standard DTD. A style-sheet is produced for the DTD, and is downloaded with the document to ensure proper rendering. SoftQuad is distributing copies of its browser to promote this concept.[7]

2.3 Macroprocessor-based systems

In this approach the master information is stored in an ad-hoc tagging format, and a preprocessor or macroprocessor tailored specifically for HTML, is used to convert the files into HTML pages. A flexible macroprocessor can greatly assist in WWW development, reducing inconsistencies and separating content from presentation and navigation structure.[8] We have developed a prototype macroprocessor system using the M4 preprocessor.

2.4 Page image systems

PDF and PostScript are popular formats used to publish information on the WWW. Both require special browsers to provide full control of the document typography.

2.5 Hypermedia-based systems

The success of the WWW has prompted the authors of some hypermedia systems such as Microcosm to adapt them to generate HTML pages. Hill et al. proposed to unify Microcosm[9] and the WWW in three ways [11]:

- Microcosm-aware WWW clients, that is enhancing clients to support Microcosm primitives.
- Generating static pages out of a Microcosm system.
- Using CGI scripts to allow the interaction of a Microcosm server with typical WWW clients. The CGI scripts would convert Microcosm requests to HTML browsable files.

In all these cases, the information does not reside on a database.

3 Modeling a WWW site

The spectacular growth of the WWW has led to a corresponding growth in ad-hoc solutions related to specific WWW problems, some of which are highlighted earlier. Categorising the information elements that constitute a WWW site and the operations performed on these elements should suggest a model that could lead to generic approaches for solving many of these problems.

The three basic types of information involved in a hypermedia system such as the WWW are: text, objects and hyperlinks. Text and objects are the fundamental units that are presented in WWW documents, while hyperlinks provide intra- and inter-document navigation.

3.1 Text

Text is ubiquitous in the WWW. Many hypermedia systems consider text only as a string of characters, but recent advances in the characterization of text have shown that structural information contained in markup languages such as those derived from SGML [10], is a fundamental component of any text-based document.

Each text element such as a paragraph or heading is tagged with an SGML tag to comply with a particular document structure or DTD.[9] Text and its associated structure should be stored separately from information about its appearance, so that the presentation can be tailored to specific environments or situations.

3.2 Objects

The remaining types of information contained in a WWW document such as images, video clips, and Java scripts can be characterized as objects. For example, consider images of paintings, which might be part of an on-line Museum on the WWW. In this particular scenario, an abstract class image could have three main methods: get_thumbnail, get_gif, and get_jpeg. However, the images we have are in tiff and PostScript format. A feasible solution is to create two subclasses for gif and jpeg, and then the translations from tiff and postscript format could be described within the methods of the object. For instance, the object image could have methods low_resolution, thumbnail,

[7] See http://www.ncsa.uiuc.edu/SDG/Software/Mosaic-/WebSGML.html

[8] We used this approach to create the WWW version of the "The University of Waterloo Undergraduate Calendar," which is composed of around 500 pages. The savings in development time were enormous compared to using plain HTML files.

[9] Document Type Definition, which formally describes the structure of a class of SGML documents.

black-and-white which are overwritten by the methods of its subclasses. Figure 1 shows the OMT diagram of the image and GIF objects with an example of their corresponding methods.

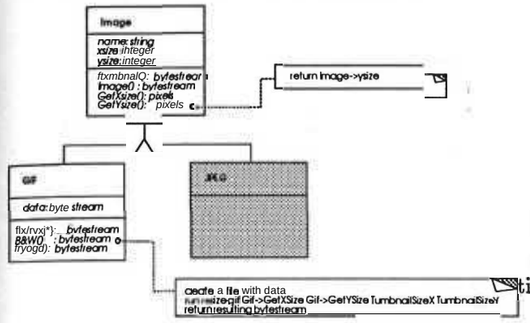


Figure 1: OMT diagram of the Image superclass

3.3 Hyperlinks

Hyperlinks are used for navigation around a WWW site, where a text segment or graphic is activated to link to a new location. Relations can be used to represent hyperlinks since the entity-relationship model has been used successfully by several authors in hypermedia system design [14, 13, 17].

3.4 Authoring Tools

The three types of entities require different tools to support creation and maintenance. For example, we can use a grammar-driven editor to process tagged text and different tools to manipulate the subtypes of objects such as sound, video or graphics.

3.5 Persistence

Text, objects and hyperlinks are persistent entities that can be modified but must last in some form for the lifetime of a WWW document. All these entities can be stored as separate files, but using databases with their many advantages for storage and retrieval is a more desirable approach.

3.6 A layered model

Since there are three distinct types of entities, and operations on these entities form groups, a layered model is a good approach to describing the relationships between entities and operations needed to construct a WWW site. Figure 2 is a diagram of the layers showing the entities in the storage layer, the operations forming the next two layers, and the completed WWW application as the final layer. This layered model is useful for describing and categorizing the various aspects of the WWW site, but the actual implementation may differ somewhat from this "ideal."

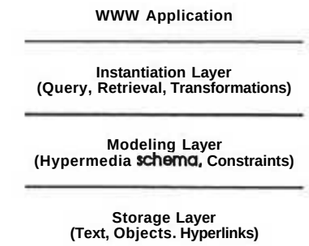


Figure 2: Layers in a hypermedia system

The storage layer

The storage layer contains the three types of entities which are stored in an appropriate database system.

The modeling layer

The modeling layer acts as the "glue" to create a unified hypermedia database. The modeling layer incorporates the following metadata:

- A hypermedia schema which is a description of the types of structures inherent in each type of entity. For example, the structure of the text could be described using a collection of DTDs, the objects through a class library, and the hyperlinks through a simple relation schema showing source-destination pairs.
- A set of constraints to maintain consistency across the database. These constraints ensure that if a textual component refers to an object or other textual component that the latter exists and is not removed from the storage layer while still being referenced.

The instantiation layer

Once the information has been retrieved and the navigational structure applied, the document still must be transformed into the appropriate appearance. This function is the responsibility of the instantiation layer which retrieves the necessary information from a repository in the form of instantiation rules and scripts. The instantiation layer includes:

- Functions to retrieve information from the database system.
- Functions to convert queries to entity retrieval commands.
- Transformation definitions. Transformations are intended to be applied to the stored entities, often in response to queries.

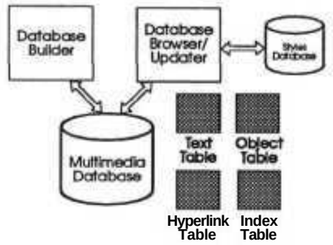


Figure 3: The base implementation

— text entities can be processed using an SGML processing language, such as **Omnimark** [7] or Jade (a DSSSL engine). For example, a number of memoranda documents tagged with SGML could be transformed by extracting the name of the author, the recipient, the date, and the first 50 characters of the text of the memo, and then presenting the result in HTML.

— For presentation purposes objects might require extra manipulation such as format conversion.

The instantiation layer could use scripts written in a language such as Perl[18] to describe how to retrieve and transform objects and how to apply the navigational information specified as hyperlinks.

Advantages of the layered model

This layered model has several advantages including:

- A clear separation of content, presentation and navigational information.
- Independence from specific storage structures allowing the designer of a system to make choices based on the available database systems and corresponding authoring tools.
- Consistency between entities ensuring the integrity of the storage model.

4 The LivePage implementation

In Section 3 we propose a layered model to describe **WWW** applications and to separate the different entities and operations. We have been developing an implementation [19, 12] of this model that satisfies the majority of the specified requirements, and uses a single relational database system. Figure 3 illustrates this implementation; the details are presented in the next sections.

4.1 The base implementation

In this implementation we initially embed text, hyperlinks and references to objects into a single WWW document, but distinguish or "separate" them by tagging conventions using an SGML-compliant tagging language. Thus, if we move to a different entity storage model, we will be able to separate the three types of information easily. Once the document is complete, we verify it against a grammar or DTD before loading the document into a single relational (SQL) database.

Each fundamental tagged structure[10] is loaded into a single field in a relational database table and given a unique identifier. There are three separate tables for the text, objects and hyperlinks. In addition, every word in the document is also placed in an index to facilitate searching when the database is browsed. Optionally, every structure tag can be placed in the same index to facilitate searching for words within specific tagged structures. The various objects such as graphics, video and sound, and links to external programs are stored directly in the object table as "blobs" with the appropriate attributes. The various tables are identified in Figure 3.

The toolkit

The LivePAGE toolkit is provided to create (the administrator), maintain (the updater), and browse and query (the browser) the database. Administrator functions, namely the verification against a DTD and creation of the database tables are described earlier in Section 4.1. We consider documents as trees [15], and the updater allows a substructure (subtree) to be removed from the database and modified or replaced. While this substructure is being changed, the corresponding section of the database can be locked in order to maintain database integrity. Since we use SQL database technology, the database can be created, updated, and browsed using SQL statements. However, the LivePAGE tools provide an interface that makes the application of the SQL statements transparent.

The administrator and updater are primarily tools for the database administrator, while the browser is a tool for the general user to examine the database. The browser supports linear browsing, following hyperlinks forward and backward, and activating objects such as audio and video clips, or links to external programs. The browser also supports queries. The queries can be Boolean or free-form where the results of the free-form query are presented in relevance order with the most relevant result presented first.

Text stored in the database can be extracted and modified using a structured text editor. Similarly objects stored in the database can be extracted using the updater and can be created or modified using appropriate authoring tools.

[10] With certain exceptions a fundamental tagged structure contains no tagged substructures.

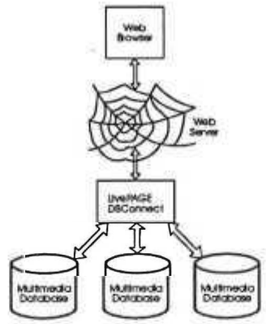


Figure 4: Access to the database over the WWW

Presentation styles

Tools such as the browser and updater allow the client to view the document stored in the database. However, the document contains only structural information. Presentation or style information is contained in a separate style database and is loaded into the browser or updater when it is invoked. The style database is indicated in Figure 3.

4.2 Connecting to the WWW

The base implementation described in Section 4.1 allows local access to documents but does not support access through the WWW. The LivePAGE toolkit contains two other mechanisms for this purpose.

The Publisher - producing a static WWW site

The Publisher is a tool that can process the database and create a set of **WWW** documents that can then be stored and retrieved from a WWW site using any of the widely available WWW browsers. In addition to providing the document content including appropriate objects and relations, the Publisher generates a "Table of Contents" (TOC) and navigation buttons as an extra navigational aid. The automatic generation of a TOC and the navigation buttons relieves the author of the WWW site of creating navigational aids, and allows users to orient themselves by returning to the TOC whenever they feel "lost in hyperspace."[11]

The generation process creates a WWW site from the database at a specific point in time. If the database changes then the "publishing" process must be repeated. The next section describes a dynamic approach to the same problem.

dbConnect - producing a dynamic WWW site

Figure 4 illustrates the architecture of a dynamic distributed document database system accessible over the WWW. Users accessing a WWW browser such as Netscape or Microsoft Explorer request a WWW page from a WWW server. Using the CGI, **NSAPI** or **MSAPI** protocol, the WWW server passes the request to the LivePAGE dbConnect module that then accesses a database of WWW pages. The specific WWW page that was requested is retrieved from the database and returned through the WWW server to the WWW browser for presentation. The LivePAGE dbConnect module is not restricted to a single database, but can retrieve WWW pages from multiple document databases as illustrated in Figure 4.

4.3 Legacy WWW sites - the Builder

There are many existing WWW sites that could benefit from the support of document databases. The LivePAGE toolkit contains a program called the Builder that can import an existing WWW site, attempt to correct structural errors, produce a TOC, and then build a document database. Once the database is constructed, all the tools previously described can be used.

5 The implementation and model - a discussion

The LivePAGE implementation retains the spirit of the layered model by maintaining the separation between the hypermedia schema and constraints, the transformations, and also the separation among the three types of entities inherent in a WWW site.

The storage model corresponds to the relational database used to handle various tables containing text, objects and hyperlinks. The hypermedia schema and constraints in the modeling layer are controlled through the DTD, where the database is checked against the DTD whenever the database is created or modified. The implementation starts with a single document which is then used to generate a set of relational tables, one for each of the three types of entities. The instantiation layer is also separate in that a style database is used for presentation in both cases. When documents are sent to WWW browsers, the style is implicit in the HTML tags.

There are a number of implementations [1, 8] that use a database structure specialized to the storage and retrieval of tagged documents. We made a practical compromise in our choice of

[11] The position in the TOC is highlighted to assist in the orientation process.

structure in that we chose a single relational database, since most organizations now support the relational model. Based on the length of time taken to adopt relational technology, they are probably a decade away from moving to other databases systems, and so the solution we propose should last a long time. Further, if organizations do switch to other database approaches later, the text, objects and relations are easily separated because of the tags.

The structure of the database used in the LivePAGE model is optimized for retrieval rather than updates, since we expect that a document will be browsed more often than modified. The organization of the internal tables of the database allows a fast retrieval of a subtree of the original document at the expense of relatively slow updates; currently, an update is implemented as a deletion and then an insertion. We are investigating better algorithms to perform updates.

There is approximately a six-fold difference between the size of the original untagged document and its equivalent database. This increase in size is caused by a number of factors including: the system tables, the indexing associated with fast retrieval of information from the database tables, and the indexing of every word in the document (with the exception of stop words).

Currently, LivePAGE is not compatible with the two emerging standards for style-sheet description of SGML documents and HTML (DSSSL [2] and CSS [5] respectively). We expect that this will be changed in future versions of LivePAGE.

The implementation, like the model, supports referential integrity. In the LivePAGE implementation referential integrity is inherent in the tagging since it is simple to verify that all the links and anchors of a document exist.

Access and update control, locking mechanisms, and rollback are all inherent in the LivePAGE implementation, since LivePage uses an SQL **relational** database as its storage model.

The LivePAGE tools provide extra navigational aids such as Table of Contents generation. Such a facility could also easily be provided in the model.

A complete WWW site is stored in a single database that can be easily moved and does not have file system dependencies.

6 Future Work

LivePage is successful in solving many of the common problems inherent in WWW site development. However, there are a number of issues that need to be addressed.

As more and more information is made available on the WWW it becomes increasingly difficult to find the information that you want. Even search tools which use ranking to return only the results that are the most relevant, are nearing their limits as to how much text they can effectively index. One technique to increase the search effectiveness is to include in the ranking algorithm, information that can be derived from the HTML tags about the structure of the document. Unfortunately, the HTML tagset is not very rich in its ability to describe the structure of all documents accurately. Augmenting HTML with additional SGML tags would be one way to overcome this problem.

Complex WWW sites do not just consist of static textual data with a few added graphics. It is often desirable to include other data, such as a stock market table, that is more dynamic and may be constantly changing. Frequently, this kind of data is also stored in relational databases. We have created a number of demonstrations of how this kind of data can be seamlessly integrated into the WWW site. Further analysis is needed to make this approach comparable to the process of building a simple WWW page.

Even the static textual data that makes up the majority of information in a WWW site is dynamic if looked at over a period of time. Frequently, a WWW site will go through several versions as it is enhanced and maintained. In some applications it is important to keep track of these different versions and to be able to recall an earlier version of the site. This form of version control could be added to the LivePAGE system.

Currently databases created for WWW sites are tagged with HTML, rather than SGML to avoid translating SGML tags to HTML tags and perhaps encountering incompatibilities. However, HTML may not adequately reflect the structure of the information, and SGML may be a better solution. However, it will be necessary to allow the database administrator to define sets of transformation rules to convert SGML into HTML. Such a capability requires further study.

The LivePAGE document database tools only support a single DTD per document, whereas a document may have several different types of structures. This feature is supported by the SGML definition but is not currently available in the LivePAGE system.

7 Conclusions

We have described a system that provides the tools necessary to create and maintain large, complex WWW sites. The system utilizes the power of SGML and relational database systems to solve many of the problems with which developers of large WWW sites are currently struggling. A clear separation is provided between the content, presentation, and navigational structure of the WWW site. This separation allows authors to focus on what they do best, writing content. On large WWW sites there will be other experts to focus on presentation and navigation. Even on smaller WWW sites, where there is only one expert, this separation can provide significant advantages by allowing the author to focus on each aspect of WWW development in turn.

References

[1] **ActiveSystems** Inc. *ActiveSystems, Reference Manual,* 1996.

[2] Sharon **Adler** (editor). *ISO/IEC DIS 10179.2:1994. Information Technology - Text and Office Systems - Document Style Semantics and Specification Language (DSSSL).* International Organization for Standardization, 1994.

[3] T. **Berners-Lee** and D. Connoly. Hypertext Markup Language - HTML/2.0. Request for Comments 1866, November 1995.

[4] T. Berners-Lee, Masinter and M. McCahill. Uniform Resource Locators (URL). Request for Comments 1738, December 1994.

[5] Bert Bos, Dave Raggett and Hakon Lie. HTML3 and Style Sheets,W3C Working Draft 10-Jul-1996, July 1996.

[6] Tim Bray. Measuring the Web. In *Proceedings of the 5th International WWW Conference,* pages 993-1005, May, 1996. W3 Consortium, Elsevier.

[7] Exoterica Corporation. *Omnimark Reference Manual version 2 Release 4.* Exoterica Corporation, 1993.

[8] EBT International. *DynaBase Reference Manual,* 1996.

[9] Andrew M. Fountain, Wendy Hall, Ian Heath and Hugh C. Davis. MICROCOSM: An open model for hypermedia with dynamic linking. In *Proceedings of the ECHT'90 European Conference on Hypertext,* Building Hypertext Applications, pages 298-311, 1990.

[10] Charles Goldfarb. *SGML Handbook.* Oxford University Press, 1990. (0-19-853737-9).

[11] G. Hill, W. Hall, D. De Roure and L. Carr. Applying Open Hypertext Principles to the WWW. In *Proceedings of the International Workshop on Hypermedia design IWHD'95,* June 1995.

[12] The Information Atrium Inc., 158 University Avenue West, Waterloo, Ontario. *LivePage Tools,* 1996.

[13] T. Isakowitz, E. A. Stohr and P. **Balasubramanian.** RMM: A methodology for structured hypermedia design. *Communications of the ACM,* Volume 38, Number 8, pages 34-44, August 1995.

[14] D. Lange. An Object-Oriented Design Method for Hypermedia Information Systems. In *Proceedings of the 28th Hawaii International Conference on System Sciences,* jan 1994.

[15] E. Mackie and J. **Zobel.** Retrieval of Tree-structured Data from Disc. In *Databases '92,* Melbourne, Australia, February 1992. Third Australian Database Conference.

[16] Jean **Paoli.** Extending the Web's tag set using SGML: Authoring new tags with Grif Symposia. In *Proceedings of the 5th International WWW Conference,* pages 1095-1103, May, 1996. W3 Consortium, Elsevier.

[17] D. Schwabe, G. Rossi and S.D.J. Barbosa. Systematic Hypermedia Application Design with OOHDM. Technical Report 30, Departamento de **Informática, Pontificia** Universidade Católica, 1995.

[18] Larry Wall and Randal L. Schwarts. *Programing perl.* O'Reilly **&** Associates, **Inc,** 1991.

[19] J. Zobel, R. Wilkinson, E. Mackie, J. Thorn, R. Sacks-Davis, A. Kent and M. Fuller. An architecture for hyperbase systems. In *1st Australian Multi-Media Communications Applications and Technology Workshop,* Sydney, Australia, July 1991.

A Framework for Complex Tokenisation and its Application to Newspaper Text

Robert Dale

Language Technology Group
Microsoft Research. Institute
School of MPCE
Macquarie University
New South Wales 2109 Australia

Robert.Dale@mq.edu.au

Abstract

A word is more than a sequence of characters between two spaces. This fact has generally been ignored in research on natural language processing; but recognising the complexity of what it is to be a word is of crucial importance if we are to add sophisticated natural language processing techniques to existing document processing applications to make them more language-sensitive.

This paper describes a framework for the tokenisation of text that tries to address this problem by providing a parameterisable approach to the tokenisation task, so that NLP components can be provided with a richer analysis of real texts. We demonstrate the ideas with application to the wide variety of word forms that appear in newspaper text.

Keywords: tokenisation, intelligent text processing, natural language processing

1 Introduction

Typical text processing tools such as word processors and text formatting tools embody very simple notions of what constitutes a word: generally, in such systems, a word is any sequence of characters bounded by spaces, or sometimes by other punctuation marks. These characterisations of wordhood are useful and appropriate where the kinds of operations that are to be performed on words are simple: for example, deleting a word, moving the cursor forwards or backwards a word, or deciding whether there is sufficient space to place a word on the current output line.

As the systems we build try to do more sophisticated things with text, these simple characterisations begin to break down. In particular, as we try to make use of techniques and ideas from research in natural language processing, we need to develop more sophisticated notions of what constitutes a word, and we need to recognise that words have internal structure that can be usefully manipulated.

We say that a system embodies LANGUAGE SENSITIVITY if it views text as more than just a sequence of characters, and takes on board linguistically motivated characterisations of the data: so, individual characters are combined into words; words are combined into sentences, perhaps with some intermediate levels of structure to indicate syntactic constituency; and sentences are combined together into paragraphs. Many current applications possess what may appear to be language sensitivity, but in general this is an illusion: it is usually the case that simple heuristics substitute for a deeper understanding.

An obvious place to look for the kinds of information and generalisations that we need for language sensitivity is in the area of natural language processing: this broad term covers technologies concerned with morphology, syntax, semantics and pragmatics, all key notions in dealing with text more intelligently. Unfortunately, much of the work in these areas is far from broad application; and more importantly from the point of view of this paper, there is a missing link that still needs to be developed. We mentioned above that words have internal structure that can be usefully manipulated. For much work in linguistics, this structure is characterised in terms of morphology: for example, the word *churches* has a base form *church* and a plural ending *es*. A great deal of work has been done in this area, and much of it is useful in the development of intelligent text processing systems; however, even before we begin to examine a word's morphology, we need to recognise that not all words are so simple, and that real texts are not as neat and tidy and well-behaved as those discussed in linguistics textbooks or used as examples in laboratory prototypes of natural language processing systems. Quite apart from important issues such as breadth of coverage, problems raise themselves much earlier.

Proceedings of the Second Australian Document Computing Symposium, Melbourne, Australia, April 5, 1997.

This is an important realisation, and one that has only recently been accepted as an issue for work in natural language processing. It has been provoked by the increasing amount of NLP research that tries to use large corpora of real texts as data; this work makes it hard to ignore the realities of text. As a result, in the last few years we have begun to see research from a natural language processing perspective that tries to say something about the characteristics of real written language: notable work in this area is Nunberg's [1990] linguistically-motivated analysis of punctuation, Grefenstette's [1994] work on tokenisation, and Palmer and Hearst's [1994] work on sentence segmentation.

The work described in this paper is in the same spirit. The topic the paper addresses is one of the first problems that has to be faced in building a bridge between text processing and natural language processing: what is a word? If we are to combine text processing techniques and natural language processing techniques for maximum effectiveness, it is precisely here that the crucial interface lies, and so it is important that we develop as robust a model of what constitutes a word as possible. The goal of the research described here is to develop an easily customisable tokeniser that can handle arbitrary text files as input, producing whatever output a client natural language processing system prefers to see.

The work described here derives from some experimental systems we have developed over the last few years; see in particular [Dale 1990; Matheson and Dale 1993; Dale and Douglas 1996]. On the basis of this research we have become convinced that there is, perhaps not surprisingly, no one answer to what should count as a word; it all depends, of course, on what task is being carried out. What we require, then, is a truly flexible approach where we can experiment with and develop different notions of wordhood. This paper describes the framework we are developing to explore these questions.

The paper is structured as follows. In Section 2, we sketch our overall approach to the problem of tokenisation. In Sections 3 and 4, we go beyond the simple notion that a word is a sequence of characters bounded by spaces and describe a framework for what we call 'universal tokenisation'; and in Sections 5 and 6 we go on to exemplify this framework in the context of an analysis of a small amount of newspaper text. Section 8 provides some concluding remarks.

2 Our Approach to Tokenisation

We begin by taking the view that a text is made up of what we will call tokens, and that tokens can be of two types, which we call word tokens and punct (for punctuation) tokens. A word token is used to represent, naturally, a word, along with any punctuation which is properly part of that word (LEXICAL PUNCTUATION). A punct token is used to represent any contiguous sequence of (non-lexical) punctuation characters in the text.

Unfortunately, the ASCII character set does not divide straightforwardly into those characters which form words and those which form puncts: some characters can belong to either, depending on the context. There are a number of these ambiguous characters, the full stop or period being the most common, since it can appear as a sentence terminator (in which case it is part of a punct) or as punctuation within an abbreviation (in which case it is part of a word). So, in example (1) below, the character string *rhino(s)* would be represented by a single word token, as would the string *eventually*, the open parenthesis immediately before the first *e* in eventually is not part of a word token, but is part of a punct token, consisting of a space and an open parenthesis and falling between the two word tokens.

(1) The rhino(s) (eventually) ate the cake.

A number of heuristics can be used in order to decide how to tokenise a text which contains ambiguous characters; when higher level knowledge is available from lexical sources and syntactic context, this can be used to disambiguate cases where there is doubt.

To enable higher-level linguistic processing to be applied to individual tokens or sequences of tokens, it proves useful to represent each token as an object that maintains information derived from the analysis of the word or punctuation sequence it corresponds to. For a given token, this structure might contain the following information:

- information regarding the syntactic category of the token, and its root form if this is different from the token itself, along with syntactic features such as number;
- information relating to the semantic type of the token, derived from a lexicon if one is available: in the context of style-checking, for example, it might be useful to know whether the word is the name of a month, or a unit of measure;
- information about the typographic form of the token: in particular, the casing of the word is

Type	Example
mixed-case-word	PhD
alphanumeric	**DEC10**
abbreviation	i.e.
	B.B.C.
	Ph.D.
ordinal-number	23rd
date	23rd December 1992
	23-12-92
os-pathname	**/home/user3/fred**
real-number	23.4
measurement	23.4 kg
latex-object	\documentstyle{article}
...	...

Figure 1: Some complex tokens

of significance, and other features such as the typeface used are important in the context of text processing.

So, for example, the first word in the sentence *Is this the best solution?* would be analysed as having the root *be*, with the syntactic features of present tense and singular number, and the typographic feature of capitalised casing.

This much is straightforward. It turns out, however, that real tokens can be quite complicated objects, with considerable internal structure beyond the morphological structure that standard natural language processing techniques can identify. Figure 1 shows some examples of the kinds of tokens we have to consider if we are to reliably process real text. Note that we have included here some tokens, such as the first example of a date token, that consist of more than one word: these are textual entities which correspond to the Text Encoding Initiative's notion of a CRYSTAL, and which for some text processing purposes are best viewed as single tokens. There is clearly a hazy Une between tokenhood in this sense and the notion of syntactic constituent that we find in the linguistics literature.

3 The Framework

To be able to process texts that contain tokens like those we have just described, it is important that we take on board the complexities of the data. After much experimentation, our current view is that tokenisation is best performed by a process that uses two separate stages.

The overall architecture is shown in Figure 2; here, everything inside the dotted Une is part of the tokeniser. The basic idea is that a stream of characters is read into the tokeniser from some external source, and then successively processed through the different internal components to produce some stream of higher level objects, which we will call TOKENS, that can be used by some **client—this** could be a parser, or some component of an information retrieval system, for example.

The system is broken down into the modules shown to provide **customisability** at a number of different levels where the **ability** to customise seems Uke a useful thing to have. The individual components have the following functionaUties.

The Bundler: The Bundler is the simplest and least intelligent part of the tokeniser. It knows only how to map the characters in the character set used into a predefined set of character classes; each character is in only one class. The **character⟶character** class mapping is defined by a control file, and so is easily changed; the Bundler uses this information to segment the input stream into **BUNDLES,** which we'll also call SIMPLE TOKENS. A bundle or simple token is simply a sequence of one or more characters from the same character class: so, for example, given appropriate definitions in the control file, any sequence of alphabetic characters might constitute a bundle, and each space character might constitute a bundle.

The Compounder: The Compounder takes as input the simple tokens provided by the Bundler and decides whether any of these simple tokens need to be coUected together into what we will call COMPLEX TOKENS. Exactly what counts as a complex token is determined by the Compounder's control file: for example, an simple alphabetic token foUowed by a simple numeric token might be put together to form a complex token. There are also what we might think of as multi-word complex tokens, or after the **TEI,** CRYSTALS. These are sequences of words and symbols, such as dates and names, that are constructed in a regular way but which are typically not catered for by a conventional natural language parser. The job of the Compounder is to build complex tokens corresponding to these crystals, packaging them up and annotating them in such a way that the client parser need not be concerned with their internal details but can **still** make use of them.

The Interface: If we want our tokeniser to be generally useful to a wide range of clients, then we cannot assume too much about the nature of the input expected by these cUents. The job of the Interface is to convert from the tokeniser's internal structures into the form of input expected by a particular cUent; again this transformation is carried out using rules specified in a control file. Thus we might expect different control files to allow the generation of lists of Lisp symbols, Usts of Prolog atoms, or complex feature structures, as required. This aspect of the tokeniser's behaviour will not be discussed further in the present paper; we wiU focus instead on the internal aspects of tokenisation.

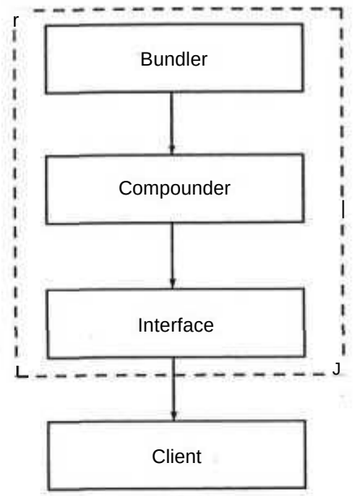


Figure 2: The overall architecture of the tokeniser

Character	Class	Iterable?
A	upper-alpha	Yes
⋮	⋮	⋮
a	lower-alpha	Yes
⋮	⋮	⋮
0	numeric	Yes
⋮	⋮	⋮
u	spaces	Yes
%	percent	No
⋮	⋮	⋮

Figure 3: A character class table

We can specify the character to character class mapping by means of a table Uke that in Figure 3; this is effectively a short-hand notation for a set of rules of the following form:

(2) a. lower-alpha ⟶ $[a\text{-}Z]^+$

b. open-paren ⟶ (

Here, the left hand side of the rule is a specification of the name of the character class (and therefore token type) in question; the right hand side of the rule is a regular expression specification of the contents of tokens of this type. These rules can then be compUed into an appropriate finite-state machine; such a **compilation** procedure would also carry out appropriate error checking and ensure, for example, that no character has been assigned to two different classes.[1]

One possible complete **character⟶character** class mapping is shown in Figure 4.[2] Note that these bundling rules have the following consequences:

1. Any word which begins with an initial capital letter foUowed by a sequence of lower-case letters wiU be viewed as a sequence of two simple tokens.

2. AU non-alphanumeric characters except spaces, line feeds, tabs and hyphens are viewed as **single-character** tokens.

The important thing to note is that these consequences are consequences of the particular bundUng rules we have specified; a different set of

4 The Individual Components

4.1 The Bundler

In what follows we wiU assume that the input characters are ASCII, although there is no reason why other character sets should not be used.

4.1.1 Specifying Bundling Rules

The means of character class specification should make it as easy as possible to specify the mapping from each character in the character set to its class. Each character class corresponds to a token TYPE. The specification also needs to have some way of saying whether a token of a particular type contains only one character or many: for example, any sequence of numeric characters constitutes a numeric token, but a open-paren token consists of a single open-parenthesis: if we find two open-parentheses in a row, we need to be able to specify that they should be separate tokens if that is the most useful way to view them.

[1] In our most recent work we have been using Yacc and Lex to perform these processes, although in the general case rather more flexible mechanisms are required.

[2] To explain the notational conventions employed here: square brackets are used to indicate ranges; symbols inside angle brackets name characters that are difficult to show in their regular form; and the use of a subscripted '+' indicates that one or more instances of the preceding character specification are required.

lower-alpha	⟶ [a-z]+
upper-alpha	⟶ [A-Z]+
numeric	⟶ [0-9]+
spaces	⟶ ␣+
line-feeds	⟶ ⟨LineFeed⟩+
tabs	⟶ (Tab)+
hyphens	⟶ _+
open-paren	⟶ (
close-paren	⟶)
open-square	⟶ C
close-square	⟶]
open-curly	⟶ [
close-curly	⟶]
open-angle	⟶ <
close-angle	⟶ >
slash	⟶ /
backslash	⟶ \
full-stop	⟶ .
comma	⟶ ,
colon	⟶ :
semi-colon	⟶ ;
exclamation-mark	⟶ !
question-mark	⟶ ?
dollar	⟶ $
tilde	⟶ ~
open-quote	⟶ ‘
close-quote	⟶ ’
double-quote	⟶ "
ampersand	⟶ &
at-sign	⟶ @
percent	⟶ %
caret	⟶ ^
asterisk	⟶ *
underscore	⟶ _
plus	⟶ +
equals	⟶ =
pipe	⟶ |

Figure 4: A sample set of bundling rules

bundUng rules can be used, without affecting the overaU approach to tokenisation.

4.1.2 Output Tokens

The tokens generated by the Bundler can be represented by feature structures Uke those shown below:

(3) a.
```
[ token: simple
  contents: ntidisestablishmentarianism
  type: lower-alpha ]
```

b.
```
[ token: simple
  contents: (
  type: open-paren ]
```

Any simple token must have three fields: a token field with the value simple, a contents field that contains a string consisting of the characters that make up the token, and a type field that specifies the character class of the token.

4.2 The Compounder

4.2.1 Overview

The Compounder's job is to put together tokens to make complex tokens: a typical instance is where the bundler has decided that, in the string *23rd*, the *23* forms one simple token and the *rd* forms another. The Compounder's rules wiU specify that these two tokens can be combined to make a complex token. So, from the tokens in examples (4a) and (4b) the Compounder might build the token in example (5).

(4) a.
```
[ token: simple
  contents: 23
  type: numeric ]
```

b.
```
[ token: simple
  contents: rd
  type: lower-alpha ]
```

(5)
```
[ token: complex
  parts: 1: [ token: simple
              contents: 23
              type: numeric ]
         2: [ token: simple
              contents: rd
              type: lower-alpha ]
  type: ordinal-number ]
```


Whereas no ambiguity was possible in the Bundler's ouptut, we now get the possibility that there may be different sequences of complex tokens: for example, a full stop might be combined with the preceding simple token to make some kind of abbreviation, but might equally be considered to be a sentence-terminating token. This requires the Compounder to make an intelligent decision on the basis of whatever sources of knowledge it has access to. In different tokenisation experiments we have used different solutions to this problem. In both the *BibEdit* [Matheson and Dale 1993] and *Editor's Assistant* [Dale and Douglas 1996] work, our tokeniser would examine the contents of surrounding tokens in order to make a decision: for example, on encountering a simple token that corresponds to a fuU stop, the compounding stage may check to see if the immediately preceding token is potentially part of an abbreviation, or whether the immediately following token could be the first word in a new sentence. A more linguistically sophisticated system might try to make use of whatever syntactic and semantic knowledge is available, but realistically such sources of information are beyond the capabilities of current systems. Another strategy, and probably the best in the longer term, is to take the view that it is *not* the job of the Compounder to decide what the correct answer is: instead, multiple parses should be produced. Also, it is possible that a number of tokens may be combined in the same way but given multiple possible interpretations: so, for example, the complex token above might also be considered to be a computer name.

Notice also that the rules used by the Compounder are potentially quite complex: ideally, in the example above, the rule needs to have some way of distinguishing ordinal-number tokens from other kinds of alphanumeric tokens. So, for example, one rule for alphanumerics might look like the following:

(6) X0 ⟶ X1 X2
(X0 type) = alphanumeric
(X1 type) = alphabetic
(X2 type) = numeric

Of course, this is not as expressive as we would like it to be: using this limited formalism for expressing compounding rules would mean that we'd need a very large number of rules to cover all the possibilities.[3]

A rule for ordinal-numbers might look Uke the following:

(7) X0 ⟶ X1 X2
(X0 type) = ordinal-number
(X1 type) = alphabetic
(X2 type) = numeric
(X2 contents) isa ordinal-ending

Here we have added the notion of contents-field type checking by means of a new operator, isa; this has the same net effect as performing lexical lookup. Ultimately, it becomes necessary to develop a type hierarchy, where types are more or less specific: so, for example, we might have alphabetic, alphanumeric and punctuated as sub-types of compound tokens, and os-pathname, hyphenated-word and date as subtypes of punctuated tokens.

[3] The formalism used here is very deliberately based on that used in the unification-based grammar formalisms popular in the natural language processing community; see, for example, Sheiber [1986].

5 Applying Tokenisation to Newspaper Text

We have described above some fairly complex machinery for breaking a text into tokens. A valid question to ask is whether this complexity is really required. This section looks at the results of an analysis of newspaper text (in particular, one issue of *The Guardian)* to identify the real variety of tokens that we have to consider. After this analysis, we provide a grammar for compounding that covers this data.

A significant proportion of the tokens found in newspaper text consist of the simple alphabetic forms; but there are a number of more complex token types too, enumerated below.

5.1 Hyphenated Compounds

A hyphenated compound is a token consisting of smaller tokens connected together by a hyphen. Identifying this structure can be important for carrying out any intelUgent processing that is required.

Figures 5 and 6 show two categories of hyphenated compounds found in the analysed text; in each case we have reproduced the entire sets detected in order to demonstrate the wide range of semantic constructs that appear, although we do not at the moment have much to say about specific semantic analyses that would be appropriate; the point is that, if we do want to perform intelUgent processing of these 'words', then we do need to have available some analysis of their internal structure.

5.2 Apostrophed Words

Apostrophed words are relatively straightforward, but once more we have to be able to decompose them appropriately to carry out appropriate processing tasks.

Possessives: *royal's, year's, Citizen's, BBC2's, MPs'* and *nurses';* a very large proportion

three-day
two-thirds
ex-inmates
ex-miUionaire
short-sighted
short-termism
well-received
sister-in-law
editor-in-chief
milk-and-water
million-year-old
three-year-old
Anglo-American
Caiger-Smith
Churchill-Coleman
Hindu-Muslim
Coca-Cola
Johnny-Come-Latelys
Serb-controlled
Italian-style
Biblical-style
Co-operation
B-Specials
UN-sponsored
US-led
al-Hariri
anti-Barre
counter-IRA
outer-London
pro-MPLA

Figure 5: Alphabetic hyphenated-compounds

of these are possessives of proper names. A case to watch out for is the apostrophed abbreviation, as in *Inc. 's.*

Contractions: *'cos, aren't, couldn't, didn't, Everywhere 's, He's, I'm, It 'll, It's, we 'll,* and *We 've.*

5.3 Number Compounds

Numbers are more than simple sequences of digits. To cater for this fact, it is useful to have a notion of NUMBER COMPOUND. As it happens, in the analysed text these are more common than simple numerics. The different subtypes we have identified are as follows:

- comma-punctuated numerics as in *250,000.*
- decimals as in *3.1* and *61.67*
- currency amounts, as in *$400, £289, £10,000, £1.8, £23.7, $12.20,* and *$116.5.*

11-14
5-2
200-400
1994-95
12-year
15-day
24-hour
100-strong
80-yard
72-year-old
18-21-year-olds
10,000-word
33,000-strong
12th-century
pre-20th
Start-2
AK-47s
MiG-23
£26-27
£100,000-plus
£l,874-a-week

Figure 6: Other hyphenated-compounds

5.4 Less Common Compounds

There are a number of other less common kinds of compound tokens. We have found the following categories to be in evidence:

alphanumerics: True simple alphanumerics—tokens consisting only of alphabetic characters and digits—are quite rare. Apart from the predictable *17th, 61st* and *194th,* we also get *1960s, DM150, M15, 28min, 01sec, 5ft,* and *5ins.*

mixed-case: Again, these are much less common than one might have thought. Examples: *MPs, CDs, McAvennie, USAir,* and *DoE.*

slashed-compounds: These are very rare: *Heath/Walker, AIRMIC/broker,* and *1993/94.*

full-stopped-tokens: These are very rare: *A., Inc.,* and *C.J..*

6 A Grammar for Compound Tokens in Newspaper Text

In the previous section we have outlined the various kinds of tokens found in our analysis of newspaper text. In this section we present a grammar for compound tokenisation that covers this data. Different compounding rules are likely to be required for other genres of text. It should also be noted that the rules were developed only to handle body text: advertisements and sports results might produce some additional token types.

We wiU make the assumption that a text consists of a sequence of tokens, and we wiU require that a text be an alternating sequence of wordand punct tokens. This means our top level rule for the decomposition of a text is as foUows:

(8) text —› {punct} word (punct word)* {punct}

6.1 A Grammar for Word Tokens

1. Words can be simple or complex, in the sense that they may consist of only one bundle, or they may consist of a number of bundles. When a word consists of a number of bundles, the alphabetic and numeric bundles may be separated by punctuation characters; where this is the case, each punctuation character may appear only once in any given complex word, with the exception of hyphens, which can appear multiple times. To deal with this, we define word tokens at the top level as foUows:

   (9) word ⟶ nohyphen-word | hyphenated-compound

2. A nohyphen-word is either simple or compound:

   (10) nohyphen-word ⟶ simple-word

   (11) nohyphen-word ⟶ compound-word

3. Words which are **simple-words** are those which contain only alphabetic or numeric characters:

   (12) **simple-word** ⟶ lower-alpha | upper-alpha | mixed-case | numeric | alphanumeric

   Of these types, only lower-alpha, upper-alpha, and numeric are provided as primitives by the BUNDLER rules specified earUer. We also therefore require the following rules:

   (13) mixed-case ⟶ (upper-alpha | lower-alpha) (upper-alpha | **lower-alpha)**$^{+}$

   (14) alphanumeric ⟶ numeric (lower-case | **upper-case)**$^{+}$

   (15) alphanumeric ⟶ (lower-case | upper-**case)**$^{+}$ numeric

   It would be useful to have a rule for **initcap-rest-lowers** tokens, but this requires more sophistication that the current rule specifications permit (it requires reference to the *length* of the bundles).

4. Words which are compound-words are of various kinds.

   (16) compound-word ⟶ number-compound

   (17) compound-word ⟶ **apostrophed-word**

5. Apart from their appearance with alphabetic characters in alphanumeric tokens, numbers often appear in conjunction with other characters. We capture aU the interesting cases with the foUowing rules:

   (18) number-compound ⟶ {currency-symbol} numeric {comma numeric} {full-stop numeric}

   (19) currency-symbol ⟶ dollar

   Note that currency amounts such as *140DM* will be identified as alphanumeric tokens.

6. apostrophed-words are handled by the following rules:

   (20) apostrophed-word ⟶ compound-word close-quote {compound-word}

   (21) apostrophed-word ⟶ **close-quote** compound-word

7. Finally, hyphenated-compounds are captured by the following rule:

   (22) hyphenated-compound ⟶ nohyphen-word (hyphens **nohyphen-word)**$^{+}$

   This gives a flat structure for multiply-hyphenated compounds, and leaves to some subsequent processing the question of whether some hierarchy should be introduced within this structure.

The complete grammar for word tokens is collected together in Figure 7.

7 A Grammar for Punctuation Tokens

Punctuation tokens are relatively straightforward. The complete set of punct tokens we permit is defined by the grammar in Figure 8. We identify three general types of punct tokens:

spacing: these are tokens whose primary purpose is to separate words; they are typically made up of combinations of space characters, but we also allow the possibiUty for a spacing token to consist simply of a sequence of hyphens.

constituent-delimiter: these are complex tokens consisting of the punctuation characters which terminate clauses and phrases, along with their associated spacing tokens.

word	→	nohyphen-word | hyphenated-compound
nohyphen-word	→	simple-word
nohyphen-word	→	compound-word
simple-word	→	lower-alpha | upper-alpha | mixed-case | numeric | alphanumeric
mixed-case	→	(upper-alpha | lower-alpha) (upper-alpha | lower-alpha)$^+$
alphanumeric	→	numeric (lower-case | upper-case)$^+$
alphanumeric	→	(lower-case | upper-case)$^+$ numeric
compound-word	→	number-compound
compound-word	→	apostrophed-word
number-compound	→	{currency-symbol} numeric {comma numeric} {full-stop numeric}
currency-symbol	→	dollar
apostrophed-word	→	compound-word close-quote {compound-word}
apostrophed-word	→	close-quote compound-word
hyphenated-compound	→	nohyphen-word (hyphens nohyphen-word)$^+$

Figure 7: The grammar for word tokens

compound-punct: these are tokens made up of parentheses along with some combination of spacing and constituent-delimiter tokens. Note that we don't consider constituent-delimiter tokens to be compound-punct tokens in this sense.

Some points to note about the grammar in Figure 8:

1. It does not include punctuation tokens that contain square, curly, or angle brackets: these are omitted for simpUcity, but would be treated in the same way as the open-paren and close-parens.

2. The treatment of open-parens allows the possibility that the open-paren may not be preceded by a space; this is required in order to deal with text-initial open-parens, but it has the consequence that word-internal open-parens wiU be labelled as non-word-internal; also, cases where the space is missing by accident will not cause the parser to fail. The same comments apply to the treatment of close-parens, except that in this case it is the following space that may not be present.

3. For simplicity, we don't specify any rules that cover backslash, tilde, at-sign, underscore, plus, equals, pipe, caret, or asterisk; these characters do not appear in the text we are using as our example.

8 Conclusions

We have described a general architecture for the parameterisable tokenisation of free-form texts, and provided details of the grammars required for one particular text type we have analysed.

The key features of the approach described here are as follows:

- The approach is completely parameterisable, so that different notions of what it is to be a word can be adopted in different contexts.
- Tokenisation is separated into two distinct steps, referred to here as BUNDLING and COMPOUNDING.
- This separation allows us to specify bundling as a deterministic, finite-state process that is cheap to implement; any more context-sensitive or inteUigent processing is localised in the compounding stage.

The results of such a process are then available for further processing by more sophisticated tools, whether these be document-processing based or natural language processing-based. For example, in some recent work carried out in conjunction with the University of Sydney, we have attached a freely available morphological analysis module to a tokeniser based on the principles described here; the results of tokenisation are then further extended with morphological information.

The result is a considerably more sophisticated notion of what it is to be a word, a step we believe to be of paramount importance in bridging the divide between text processing and natural language processing.

Acknowledgements

The ideas expressed here have benefitted from discussions with Shona Douglas, Jason Johnston, Chris Manning, and CoUn Matheson; all errors remain the author's own.

punct	→	spacing | constituent-delimiter | compound-punct
spacing	→	space | line-feeds {space} | tabs | line-feeds {tabs} | hyphens | space hyphens space
constituent-delimiter	→	(full-stop | comma | exclamation-mark | question-mark | colon | semi-colon) spacing
compound-punct	→	{spacing} (open-paren | open-quote)
compound-punct	→	(close-paren | close-quote) {(spacing | constituent-delimiter)}

Figure 8: Rules for Punct tokens

References

R Dale **(1990)** A Rule-based approach to Computer-Assisted Copy Editing. In *Computer Assisted Language Learning,* 2, pp59-67.

R Dale and S Douglas (1996) Two Investigations into InteUigent Text Processing. Pages 123-145 in *The New Writing Environment,* edited by Mike Sharpies and Thea van der Geest. Springer, London.

G Grefenstette (1994) *Explorations in Automatic Thesaurus Discovery.* Kluwer Academic Publishers, Dordrecht.

C A Matheson and R Dale (1993) BibEdit: A Knowledge-Based Copy Editing Tool for Bibliographic Information. In E S Atwell (ed), *Knowledge at Work in Universities: Proceedings of the Second Annual Conference of the Higher Education Funding Councils' Knowledge Based Systems Initiative.* Cambridge, December 1993.

G Nunberg (1990) *The Linguistics of Punctuation.* CSLI/University of Chicago Press.

D Palmer and M Hearst **(1994)** Adaptive sentence boundary disambiguation. In *Proceedings of 4th ACL Conference for Applied Natural Language Processing,* Stuttgart, October 1994.

S M Shieber (1986) *An Introduction to Unification-based Approaches to Grammar.* The University of Chicago Press, Chicago, IlUnois.

Applying a Generic Conceptual Workflow Modeling Technique to Document Workflows*

Wasim Sadiq *Maria E. Orlowska*

CRC for Distributed Systems Technology
School of Information Technology
The University of Queensland
Qld 4072 Australia

{wasim,maria}@dstc.edu.au

Abstract

The workflow technology is emerging as an appropriate platform for the automated coordination of business activities. The documents represent the primary medium of business communication. Almost all kinds of business activities have some associated documents. The workflow management systems could be applied to coordinate the flow of business documents. However, before this automated document workflows could be implemented, we need to apply some conceptual modeling methodology to capture, analyse, and describe the role and flow of documents in a business process. In this paper, we present a generic conceptual workflow modeling technique that should be applicable to all kinds of workflows. The document workflows represent a specialized application of workflows. We identify basic characteristics of document workflows and discuss some issues that should be targeted during the modeling process. We also apply the proposed conceptual modeling technique to model an example document workflows application for handling postgraduate admission process of a university.

Keywords conceptual modeling of workflows, document workflows.

1 Introduction

The documents represent the primary medium for business communication and play an important role in the effectiveness of business processes. The uses of documents in organizations range from inter office communication, e.g., memorandums, to mission critical applications, e.g., purchase orders. Even before the advent of computers, the organizations had been trying to improve the efficiency of document flows by designing effective physical means.

In early days of business computing, the computers were mainly used in mission critical data processing activities. However, they are now extensively utilized in almost aU areas of a business.

One of the major activities in any business is to monitor and control the flow of business documents and to ensure that the proper actions are performed on the basis of the information contained in the documents by the right persons and at the right time.

The coordination of business activities has historically been performed manually. In recent years, the possibility of automating the coordination of execution of these different processing activities has been explored and called workflows. The workflows represent the organizational flow of information from one processing entity, either manual or automated, to another. The processing entities use this information to accomplish assigned tasks. These tasks take some information from the preceding tasks, perform some work based on the information received using the services of the assigned processing entities, and proceed to the next tasks in the workflow. The workflow management systems partially or fully take over the responsibility for coordinated execution of tasks from human coordinators. The execution coordination of tasks and processing entities is accomplished by enforcing procedural constraints.

In literature, there is no clear cut classification of workflows. The trade press classifies workflows into three categories: ad-hoc, administrative, and production on the basis of repetitiveness, predictabiUty, control, and functionality requirements of tasks and workflows. Another classification is made on the basis of the processing entities: system-oriented and human-oriented [5]. The application-centric classification is another way of differentiating workflows applications. Using this approach, we classify workflow applications by high-level appUcation characteristics. The document workflows are classified through this approach and include workflow appUcations that manage the coordination of business documents. In document workflows the primary objects for task operations are documents and the processing entities are mainly humans.

The Workflow Management Coalition was founded in 1993 with the mission to promote the use of workflows through the estabUshment of standards for workflow technology. The coaUtion has proposed a reference model for the development of workflow management systems [13]. The reference model assembles the generic components of workflow systems into six groups. One of the groups contains process definition tools that may be used to analyse, model, and describe business processes. The work presented in this paper defines a set of workflow modeling objects and constructs that conform to the specifications of process definition tools group of the workflow reference model.

Before a workflow management system can be used to manage workflows, we apply process definition tools and techniques to model the workflow processes. This modeling information about the procedural rules is stored in a workflow repository. The workflow management systems make extensive use of this workflow repository in their operations. Using the stored process definitions in the repository, workflow management systems create and execute workflow instances and coordinate the interactions among the tasks within a workflow instance.

The rest of the paper is organized as foUows. In Section 2, we briefly describe our proposed conceptual workflows modeUng technique. In Section 3, we apply the modeUng technique to Ulustrate an example document workflows application. We describe the characteristics and requirements of document workflows in Section 4. In Section 5, we identify the limitations of our modeling language. We conclude the paper in Section 5.

2 Conceptual Modeling of Workflows

The objective of a conceptual workflow model is to produce high-level specifications of workflows independent of the workflow management software. A few workflow modeUng techniques have been proposed and reviewed in the Uterature targeting specific workflow aspects [3][5][7][8][10][11][12].

The primary objective of a workflow management system is to coordinate the execution of activities or tasks of an organization. Correspondingly, a workflow modeUng methodology should cover the techniques and tools to capture, analyse, and describe different aspects of tasks and their coordination. The requirements for workflow specifications may vary for different appUcation areas.

The motivation of the work presented in this paper is to identify the specific characteristics of document workflows and apply our proposed modeUng technique to model an example document workflows application.

In this section, we briefly present the objects and constructs used in our workflow graphical modeUng technique. A more detailed specification of our workflow modeling methodology can be found in [12].

2.1 Workflow Modeling Objects

Most of the information modeUng techniques include graphical representation that enhances the understanding of the models. A workflow specification could also be represented using graphical objects. The workflow specifications model proposed in this paper includes four types of objects: task, condition, synchronizer, and flow. The flows are used to Unk the first three types of objects together to build workflow specifications. Figure 1 shows the graphical representation of these objects in our model.

Figure 1: Workflow ModeUng Objects

Task

A *task* is the primary workflow modeUng object and represents the work to be done to achieve some given objectives. The aim of workflow modeling is to capture the coordination requirements for performing a set of tasks for a given business process. All other modeling objects except the task are internal to the workflow management system and are used to specify the rules and constraints for the coordination of workflow execution. Tasks have many properties representing their different aspects Uke temporal, transactional, user-oriented, etc. The modeUng of internal task structures is a speciaUzed area of workflows and is of vital importance for the design and operations of workflow enactment services. In this paper, however, we concentrate on the workflow modeUng at a higher level where we emphasize on the overaU coordination of tasks and do not consider the internal working of tasks.

Condition

A *condition* is appUed to represent alternative paths in workflow specification depending on an externally provided condition value.

Synchronizer

At certain points in workflows, it is essential to wait for the completion of more than one execution path before proceeding ahead. A *synchronizer* is used for this purpose and it simply waits until all the incoming flows have been activated.

Flow

A *flow* defines the connection between any two objects, other than flows, in the workflow. It shows the flow of information as control and data parameters from one object to another. By connecting workflow objects with the help of flows, we build directed graphs of workflow specifications where flows represent edges; and tasks, conditions, and synchronizers represent vertices.

* The work repotted in this paper has been funded in part by the Cooperative Research Centres Program through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia.

Proceedings of the Second Australian Document Computing Symposium, Melbourne, Australia, April 5, 1997.

2.2 Modeling Workflow Constructs

Using the workflow modeUng objects, we can create workflow specifications by building and joining different constructs. In this section, we define and explain these constructs. Figure 2 **illustrates** a workflow graph where the graphical representation of these constructs has been identified.

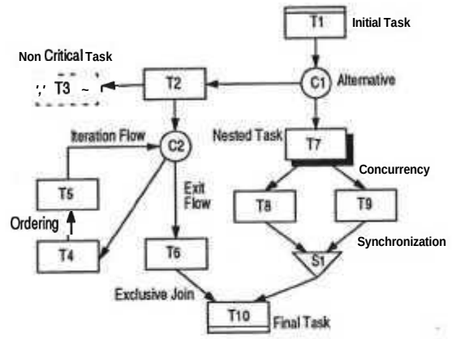


Figure 2: Workflow Modeling Constructs Usage

Ordering
The ordering is the most basic construct of workflow modeUng. It specifies, by connecting modeling objects with flows, the execution order of the tasks in a workflow.

Alternative
The alternative construct is used to model two or more mutually exclusive alternative paths in a workflow. It is constructed by attaching two or more outgoing flows to a condition object The condition is a primitive object in our model and depends on external parameters. It takes a set of control and data parameters and a condition value from the preceding task at run time. On the basis of the condition value, it selects one of the alternative workflow execution paths for a given instance of the business process by activating one of its outgoing flows. It is essential in our model that the condition is exclusive and complete. The *exclusive* characteristic ensures that only one of the alternative paths is **selected.** The *complete* characteristic guarantees that, in aU instances of the workflow that invoke the condition, one of the alternatives shall be selected.

Exclusive Join
The exclusive join construct is opposite to the alternative construct It is applied to join two or more mutually exclusive alternative paths into one path. It is constructed by attaching two or more incoming flows to a task or condition object.

Concurrency
The concurrency construct is used to represent two or more concurrent execution paths within a workflow. It is modelled by connecting two or more outgoing flows to a task or a synchronizer.

Synchronization
The synchronization is opposite to concurrency just as exclusive join is to alternative. It uses the synchronizer object and has two or more incoming flows and one or more outgoing flows. The outgoing flows are not activated until aU the incoming flows have been activated. This construct lets us model the situations where we want two or more concurrent execution paths to complete before proceeding.

Iteration
We need the iteration construct to model the repetition of a group of tasks within a workflow. However, the iteration construct results into a cyclic representation of workflow graphs and thus adds complexity. The iteration is modeled through a condition object. As long as the condition selects an iteration path, a sub graph of the workflow is repeated. Within each iteration construct, at least one of the flows exists that introduces iteration. We call such a flow an iteration flow. The *iteration flow* connects the last object in the iteration to the first object of iteration. Furthermore, at least one of the flows in iteration construct exists, caUed *exit flow,* that connects to a path outside the iteration through the condition object

Start / Stop
The initial and final tasks are used to represent the start and stop constructs of workflow specifications. All workflows have at least one initial task and one final task. These two types of tasks represent the alternative initiation and termination of the workflow execution. Even if a workflow has more than one initial and final task, only one initial and one final task would take part in a particular workflow execution instance. Generally, an initial task has one or more outgoing flows but no incoming flows and a final task has one or more incoming flows but no outgoing flows. After a final task completes its execution, the workflow execution is considered successful. By default aU the tasks without an incoming flow are treated as initial tasks. However, the reverse is not true for final tasks. A task without an outgoing flow could either be a final task or a non critical task. In our model, a workflow may contain disconnected workflow subgraphs. More than one subgraph allow us to model mutually exclusive alternative user selected execution paths for a business process.

Nesting
The nesting construct simplifies the workflow specifications through abstraction. Using this construct, we can encapsulate a workflow specification into a task and then use that nested task in other workflow specifications. For each execution of a nested task, the underlying workflow is executed. It is important to perform this nesting in a logical way following a modular approach otherwise it could add complexity to the workflow specifications rather than simplification.

Internal Null Tasks
The Workflow Management Coalition [13] identifies four primary workflow control structures: **OR-Split,** OR-Join, **AND-Split,** and AND-Join. These are represented in our model through alternative, exclusive join, concurrency, and synchronization constructs respectively.

We have two graphical objects to model the alternative and synchronization constructs: condition and synchronizer. However, the exclusive join and concurrency are represented simply by directly connecting flows to the objects. Two or more incoming flows to tasks and conditions represent exclusive join. Two or more outgoing flows from tasks and synchronizers represent concurrency. This simpUfied approach requires minimum number of modeUng objects. Nevertheless, in certain cases, it requires the use of *null internal tasks* whose only purpose is the coordination of flow and compliance to the syntactical correctness criteria of workflow constructs. Figure 3 shows a few cases where the use of internal nuU tasks is necessary. The internal null tasks have been highlighted in the examples.

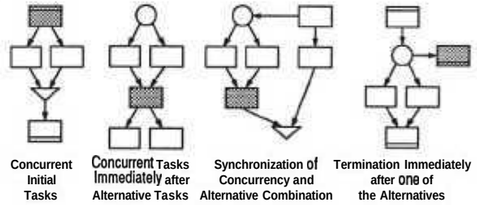


Figure 3: Use of Internal Null Tasks

Identifying Constructs
Table 1 summarizes the identification of constructs on the basis of object types and the number of incoming and outgoing flows connected to them.

Table 1: Identification of Constructs

Objects	Incoming Rows			Outgoing Flows		
	None	1	>1	None	1	>1
Task	I	**O**	J	F,N	O	**C**
Condition	X	0	J	X	X	A
Synchronizer	X	X	**S**	F	**O**	**C**

O: Simple Ordering, A: Alternative, J: Exclusive Join,
C: Concurrency, S: Synchronization, I: Initial Task,
F: Final **Task,** N: Non Critical Task, X: Illegal

3 Document Workflows: A Case Study

In this section, we apply our workflow modeling technique to a document workflows application for the postgraduate admission process of a university. In this workflows appUcation, admission documents go through different stages and decisions are made by appropriate processing entities. The workflows model presented here does not deal with the implementation details of the application.

With the help of nesting construct, we model the workflows appUcation using seven workflow graphs.

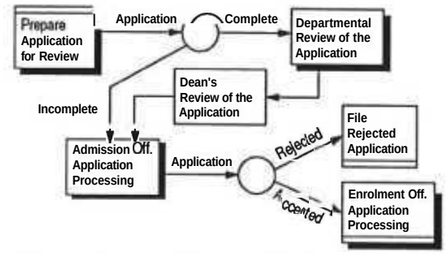


Figure 4: Postgraduate Application Processing

Figure 4 shows the top level workflow for the whole admission process. It contains four nested tasks, each of which is decomposed into independent workflows. An instance of this top level workflow is created for each appUcation for postgraduate admission. The first task of the workflow is to prepare an application file for review. After the application file is successfuUy prepared, the workflow management system would forward it for departmental review. The next task in the workflow is a review of the application by respective Dean. He or she, after the review, sends the application back to the admission office for final processing. If the appUcation is accepted, the file would be sent to the enrolment office for their processing. Otherwise, it would be filed as a rejected appUcation.

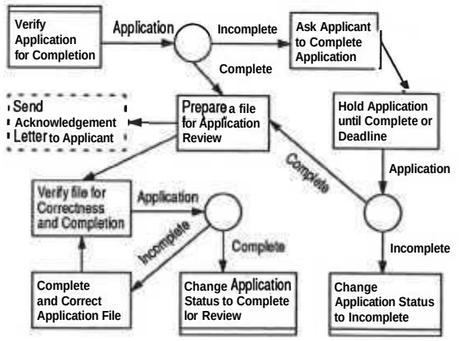


Figure 5: Prepare Application for Review

Figure 5 represents the decomposition of the first task of the workflow in Figure 4. If the application is incomplete, the admission office would contact the applicant for completing the application and then wait untU the appUcant has completed the application requirements or the response is not received by deadUne. If the appUcation is complete, a file for application review would be prepared by one of the admission

office staff. To ensure that the file has been prepared correctly, the admission supervisor would verify its contents. The file preparation activity would be iterated until the preparation of application file is correct and complete. At the end of this workflow, the appUcation file would have either a complete or an incomplete status depending on the path it would take.

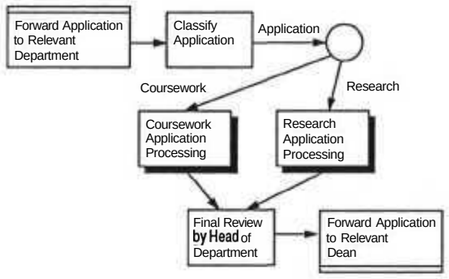


Figure 6: Departmental Review of Application

In Figure 4, a complete application is sent for departmental review. Figure 6 models the decomposition of departmental review **task.** Each applicant applies for enrolment in a specific department. The first task in this workflow forwards the appUcation to the respective department The department would classify the application in two categories, coursework or research, and process it accordingly. The appUcation processing is once again represented through another level of nested tasks. After the application processing is completed by the department, the head of the department would perform a final review of the appUcation and forward the application to the respective Dean.

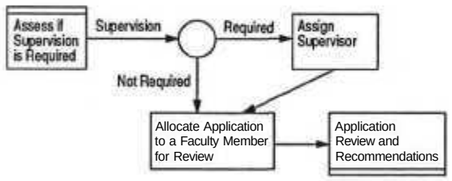


Figure 7: Coursework AppUcation Processing

If the appUcation is for **coursework,** the workflow in Figure 7 would be used for processing. Depending on the appUcation requirements, the department would decide whether the supervision of the appUcant is necessary. If it is, an appropriate supervisor would be assigned. Then one of the department faculty member would be assigned to review the application. The final task of this workflow is the appUcation review and recommendations by the assigned faculty member.

If the application is for research in Figure 6, the workflow in Figure 8 would be initiated. The department would assess, looking at the appUcant's background and proposed research, if some appropriate supervision is **available** in the department. If it is not, the appUcation would be marked as unacceptable.

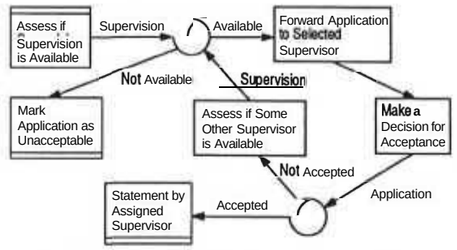


Figure 8: Research AppUcation Processing

It is possible that more than one faculty member are available for the proposed research area. The appUcation would be sent to each one of them using the iteration construct **until** one of them accepts the appUcation. If that happens, the assigned supervisor would make his acceptance statement If none of the supervisors are interested in supervising the applicant, the application would be marked as unacceptable.

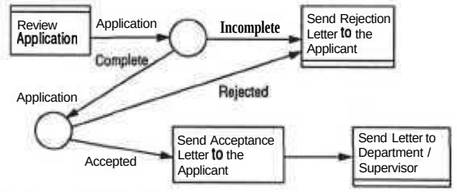


Figure 9: Admission Office Application Processing

The workflow in Figure 9 represents the decomposition of the admission office application processing task of the top level workflow. This workflow could be initiated in Figure 4 from two alternative paths. If the application is incomplete, there would not be any departmental and dean review. In this case, the first task in this workflow would directly proceed to the task for sending a rejection letter to the appUcation. Otherwise, it would perform the other tasks depending on the application status.

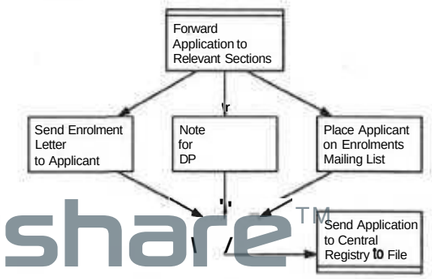


Figure 10: Enrolment Processing

The workflow in Figure 10 models the enrolments office processing for an accepted appUcation. In this workflow, three tasks would be initiated concurrently. After aU of them have finished their processing, the application would be sent to central registry to file.

The use of nesting construct simpUfies the modeling process and enhances the understanding of a workflows appUcation. We have combined **all** of the above workflows into one large workflow in Figure **11** to demonstrate the difficulty of comprehending a large workflow. It is also easier to verify the correctness of small nested workflows than one large workflow.

4 Document Workflows

In this section, we look at some workflow classification schemes in context of document workflows. We also identify some general characteristics and modeUng issues applicable to document workflows appUcations.

The workflows could be classified into three categories: ad-hoc, administrative, and production on the basis of repetitiveness, predictabUity, control, and functionaUty requirements of tasks and workflows [5].

The *ad-hoc workflows* do not have a very weU-defined pattern for moving information from one processing entity to another. The ad-hoc workflow management systems typicaUy require extensive human coordination for their operations. The execution path may be different for different workflow instances and is defined at the time a workflow is performed.

The *administrative workflows* are **similar** in nature to the ad-hoc workflows except that the workflow execution is repetitive and predictable and it foUows a well- defined execution path. Both ad-hoc and administrative workflows are usually not mission critical and do not interact with the organizational information processing systems. The primary use of such workflow management systems is to coordinate the flow of office information among people in the form of documents, reports, emails, etc.

The *production workflows* involve the coordination of organizational information processing systems that are usually based on database management systems. These workflows have well-defined procedures for the repetitive coordination of business activities and may span over several heterogeneous information systems of the organization. The production workflow management systems are more complex than ad hoc and administrative workflow management systems. They must have extensive features to define the internal task structures, control the execution of tasks involving different types of processing entities, and support reliable **failure** recovery.

The *transactional workflows* faU under the definition of production workflows. They extend the basic workflow model by introducing well-tested transactional features of the transaction management systems.

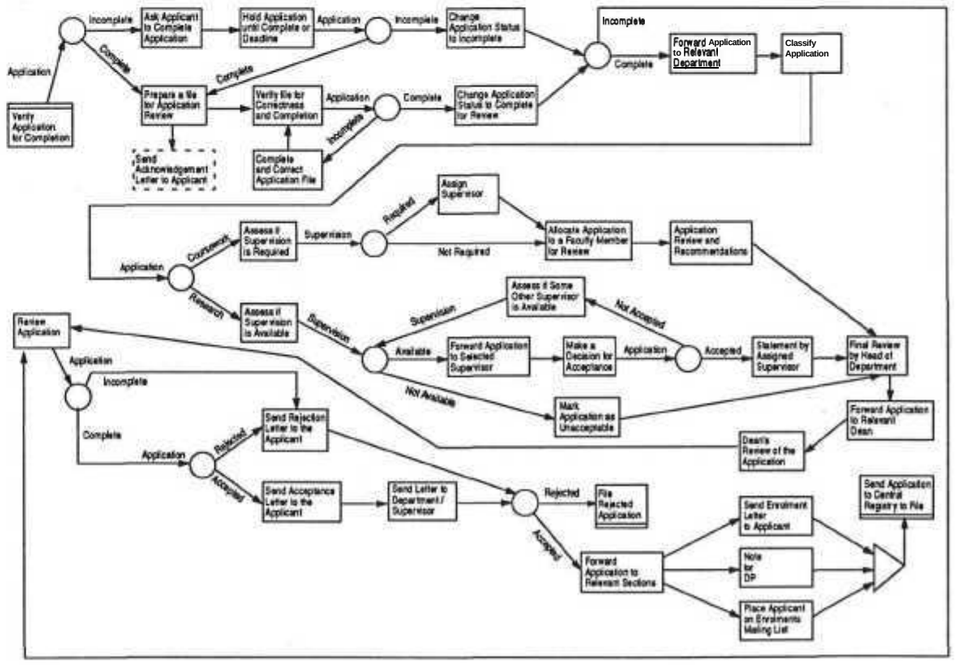


Figure **11:** Postgraduate Application Processing

A document workflows application could faU under any of the above three categories depending on the application requirements. For example, a workflows application for the preparation, review and authorization of business contracts in an organization could either be an ad-hoc or an administrative document workflows appUcation. The workflows application is ad-hoc if the flow of each contract document is different and is defined by humans at run time. However, if the flow of contract documents is known before hand and is similar for aU contracts, the workflows appUcation is administrative.

The postgraduate admission workflows presented in Section 3 could be implemented either as an administrative or a production workflows appUcation since aU the steps are well-defined and predictable.

Suppose we do not have a computerized admissions processing system and paper files are used to assess and make decisions about the eligibility of applicants for admission. In this case, if we use a workflow management system to coordinate the admissions process, its primary objective would be to monitor the status of admission applications. Since it would not be interacting with any information systems and database management systems for admissions processing, it would represent an application of administrative workflows.

However, if we have a computerized admissions processing system where academic details of apptcants are maintained in a database. The relevant documents Uke transcripts, degrees, reference letters, etc., are stored electronically. All the assessments and decisions are also made on-line and updated in the database. Then the same admissions process would represent a production workflows application.

Another workflows classification is made based on the type of processing entities: *system-oriented* and *human-oriented* [5]. The tasks in documents workflows are primarily performed by humans, therefore, they faU under the human-oriented workflows classification. This classification, however, is very general.

We could also classify workflows by applying *application-centric* approach. Using this approach, we look at very high-level requirements of a workflows application and identify its overall character. Primarily, there are four building blocks of a workflows appUcation: objects, tasks, performers, and constraints. The overaU characteristics of these four components identify the nature of the application. We use the keyword *work* from the workflows to define the four building blocks:

- *Work is performed on Objects*. An object could represent any entity of interest. In postgraduate admission workflow, the admission appUcation form represents a workflow object.
- *Tasks specify the work to be done*. The work specified by tasks is generaUy performed on objects or on the basis of information contained in objects. For example, the review of the application form by the appropriate Dean is one of the tasks in postgraduate admission workflow.
- *Performers carry out the work*. Tasks cannot perform the work themselves. They need the services of performers. For example, a Dean is the performer of the task to review the admission application form and make a selection decision.
- *Constraints control correct execution of the work*. Tasks specify what to do and performers carry out that work. The constraints ensure that the work is performed correctly. For example, one of the constraints of the Dean's review of appUcation form task could be to complete the review within two working days after receiving the application form.

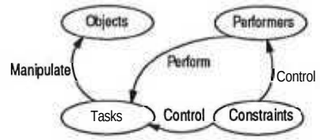


Figure 12: Workflows Building Blocks

For each business process, we could identify the above four components. GeneraUy, the document workflows have foUowing high level properties for these building blocks:

- *Objects:* primarily sets of documents.
- *Tasks:* addition to or modification of the contents of documents; decisions and actions based on the information in documents.
- *Performers:* mainly humans.
- *Constraints:* mostly human-oriented.

Looking at the above components, we can define the document workflows as follows: "The *document workflows* specify the automated coordination of a business process where humans work on related business documents and take actions foUowing a set of procedural constraints to achieve some business objectives."

The workflows appUcations, in general, have certain characteristics. Some of them apply only to document workflows. A workflows modeling methodology should target these properties during the modeling process. In the following paragraphs, we identify some of these properties and discuss them in context of document workflows. We also refer to the postgraduate admission application presented in the previous section for explanation.

Tasks Work Mostly on Documents

A primary difference between document workflows and other kinds of workflows is the type of objects they work on. The workflows are applied to coordinate a set of tasks that are performed following a set of procedural constraints. These tasks operate on objects. The tasks take decisions or transform objects from one state to another. The objects could represent entities like documents, databases, computer programs, etc., that are required by the business process to achieve its objectives.

The type of objects document workflows are based on are mainly documents, therefore, they inherit the generic properties of documents. The document workflows have to deal with two types of documents: structured and unstructured. The structured documents follow the same pattern for all of their occurrences and could be mapped into a database table with predefined structure. The unstructured documents, as the name suggests, contain dissimilar information structure for different occurrences. This differentiation is not very significant if the workflow management system is coordinating business process based on manual documents. However, for a computerized document processing system, the workflow management system needs to differentiate between these two kinds of documents.

For example, an appUcation file for the postgraduate admission contains the appUcation form, degrees, transcripts, reference letters, etc. The application form represents a structured document. Such a form would normaUy have predefined questions with specified spaces for applicants' replies. The information in this form could be mapped directly into a database table with fields representing the applicant's replies to the questions in the application form. However, transcripts, reference letters, degrees, and resume may have different structure for different applicants. In this case, we could scan the documents as images and store them as images or converted text in the document database. GeneraUy, a flexible storage poUcy is required for unstructured documents. The implementation approach for the operations of a document based business process directly affects the implementation of the corresponding workflows application.

Primary Performers are Mainly Humans

In workflow management systems, most of the tasks require both humans and computers as their performers to carry out their operations. We could classify the performers for a particular task in two categories: primary and supporting. The primary performers are directly responsible for carrying out task operations with some help from supporting performers. For most of the tasks in document workflows, humans represent primary performers and computers represent supporting performers. For example, in postgraduate admission workflow, most of the tasks represent some action, review, or decision by humans based on the information in application file. The computers responsible for executing programs that provide this information to humans represent supporting performers.

Management of Authorization Levels

In some document workflows, during the flow of documents from one person to another, certain authorization is required. In case of paper documents, the signatures and stamps are used for this purpose. The fully computerized document workflows must also include certain features to replace manual authorization. For example, in postgraduate admission workflow, we require application review by the Dean after completing the departmental review. The workflow management system would wait for the Dean's decision regarding the selection or rejection of admission application. The workflows application for this process must have the ability to ensure that only the authorized Dean makes the decision on appUcation. The system should also maintain some sort of electronic signature for later verification if required.

Distributed Task Allocation Among Groups

Another important feature of document workflows is distributed aUocation of tasks to a group of performers. As discussed earlier, tasks use the services of performers for carrying out their operations. In a workflows appUcation that involve a large number of humans as performers, effective task aUocation policies are even more important. For example, at one stage, the coursework postgraduate appUcation is sent to a faculty member for review. In each department, there would be a group of faculty members who could perform this task. There could be several considerations for aUocating an application to a faculty member. For example, we may want to maintain a balanced workload among faculty members for appUcation review. In this case, the workflow management system would allocate the task to a faculty member who is reviewing the least number of appUcations. These task allocation policies could be very complex and may involve many constraints. It is important to capture and analyse the task allocation policies during the modeUng process.

Complexity of Temporal Constraints

The specification of temporal constraints is an important aspect of workflow modeUng. Nevertheless, they significantly escalate the complexity of workflow specifications. For example, we may define a deadUne for the completion of a particular task. We may also require that a particular task executes only at a specified time of the day or waits for a specific delay after activation. A workflow without temporal constraints starts executing immediately after activation, finishes its execution as soon as possible, and does not have any deadline for completion.

In postgraduate admission example, we may require that an application must be processed within a specific time period. Then, we could divide that time among different activities. This means, untimely completion of a particular task would affect the timely completion of the proceeding tasks. However, there are several constraints like workload, availability and efficiency of staff, holidays, etc., that may result into delays in the completion of certain tasks. The identifi-

cation of these temporal constraints and associated bottlenecks during the modeUng process is very important for workflows involving a large number of performers and expected workflow instances. Another important aspect related to temporal constraints is the predictability of the life span of workflow instances.

Obligatory Constraints

The workflows with obligatory constraints must be able to handle any number of instances. These constraints, along with temporal constraints, add to the complexity of workflows applications. For example, in an immigration department, we could specify a quota for new immigrations for each year. In such a case, we could precisely plan ahead the schedule for immigration appUcation process. It is not an obligation of the department to process each application they receive. They could just process as many appUcations as required to meet the desired **quota.** If that quota is met earlier in the year, no more immigration applications would be processed. On the other hand, it is an obligation of the university to process all the applications they receive within a specified time frame. They may select only a limited number of appUcants, but they cannot teU an applicant that his or her appUcation cannot be processed because of some administrative difficulties. The efficiency and flexibfiity of obligatory workflows applications are very important

5 Extending the Modeling Language

We are working on a generic workflow modeling methodology for process definition that should be apptica**ble** to all kinds of workflow applications. The methodology, when completely developed, should cover several important aspects of workflow appUcations. The primary objective of a workflow management system is to coordinate the activities or tasks of an organization. Correspondingly, a workflow modeling methodology should cover the techniques and tools to capture, analyse, and specify the following different aspects of tasks and their coordination:

- graphical modeling objects and constructs;
- task characteristics and properties;
- task structures;
- data exchange properties among tasks;
- inter task dependencies;
- task execution and scheduling constraints;
- task failure and recovery management;
- exception management;
- criteria for assigning tasks to processing entities;
- correctness and reliabUity verification;
- evolution management;
- repository specifications.

The workflow specifications may vary for different application areas. For example, we may not want to capture the task failure and recovery management properties for ad hoc workflows.

In this paper, we have presented a graphical workflow specification language that **facilitates** the modeUng of workflow coordination through modeling objects and constructs. However, there are some other important aspects as discussed in previous section, e.g., temporal and obUgatory constraints, that should also be targeted during the modeling process. The **abil**ity of our proposed language for capturing such constraints is limited. However, we are currently targeting these issues and extending the capabilities of our workflow modeUng language.

6 Conclusions

In this paper, we have introduced a graphical conceptual modeling technique for workflows and applied it to model an example document workflows application for postgraduate admissions process of a university. The graphical modeling objects in **our** modeUng technique include tasks, conditions, synchronizers, and flows. Using the four modeling objects, we have identified the following modeling constructs: ordering, alternative, exclusive join, concurrency, synchronization, iteration, **start/stop,** and nesting. These constructs are used to specify rules and constraints for workflow execution.

The document workflows represent a specialized form of workflows applications. We have presented a few workflows classification schemes and have shown that they do not clearly differentiate document workflows from other kinds of workflows. One way to distinguish document workflows is to use an application centric classification. The workflows are built on four building blocks: objects, tasks, performers, and constraints. Using the application centric approach, we look at these four components of a workflows application and identify its nature at a very high level. We have also presented general characteristics of document workflows and discussed some issues that should be targeted during the modeling process.

The language presented in this paper **facilitates** high level conceptual modeUng of workflow coordination. However, the abUity of our proposed language in capturing coordination constraints like temporal, obligatory, allocation, etc., is limited. We are currently extending our modeling language to capture such constraints.

References

[1] P.C. Artie, M.P. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and Enforcing Intertask Dependencies. *Proceedings of the 19th VLDB*, Dublin, Ireland, 1993.

[2] Y. Breitbart, A. Deacon, H.J. Schek, A. Sheth, and G. **Weikum.** Merging **Application-centric** and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows. ***Sigmod*** *Record*, 22(3), September 1993.

[3] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual Modeling of Workflows. In M.P. Papazoglou, editor, *Proceedings of the OOER'95, 14th International Object-Oriented and Entity-Relationship Modeling Conference*, volume 1021 of *Lecture Notes in Computer Science*, pages 341-354. **Springer-Verlag,** December 1995.

[4] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. In *Proceedings of the 15th International Conference on Conceptual Medelling*, ER'96, Cottbus, Germany. Lecture Notes in Computer Scence. Springer Verlag, October 1996.

[5] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process ModeUng to Workflow Automation Infrastructure. *Journal on Distributed and Parallel Databases*, **3(2):119-153, 1995.**

[6] A.H.M. ter Hofstede, M.E. Orlowska, and J. Rajapakse. Verification Problems in Conceptual Workflow Specifications. In Proceedings of the 15th International Conference on Conceptual MedeUng, ER'96, Cottbus, Germany. Lecture Notes in Computer Scence, pp. 73-88, Springer Verlag, October 1996.

[7] D. Kuo, M. Lawley, C. Liu, and M.E. Orlowska. A General Model for Transactional Workflows. *Proceedings of the International Workshop on Advanced Transaction Models and Architecture*. Goa, India, August 1996.

[8] M. **Kamath** and K. **Ramamritham.** Modeling, Correctness & Systems Issues in Supporting Advanced Database Applications Using Workflow Management Systems. Computer Science technical report 95-50, University of Massachusetts, Massachusetts, June 1995.

[9] C. Mohan, G. Alonso, R. Gunthor, and M. Kamath. Exotica: A Research Perspective on Workflow Management Systems. *Data Engineering*, Vol. 18, No. 1, March 1995.

[10] J. Rajapakse and M.E. Orlowska. Towards a Graphical Transactional Workflow Specification Language. *Proceedings of Australian Systems Conference*. September 1995.

[11] M. Rusinkiewicz and A. Sheth. Specification and Execution of Transactional Workflows. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond*. Addison Wesley, 1994.

[12] W. Sadiq and M.E. Orlowska ModeUng and Verification of Workflow Graphs. Technical Report No. 386, Department of Computer Science, The University of Queensland, **Australia,** 1996.

[13] Workflow Management Coalition (1996) The Workflow Management Coalition Specifications - Terminology and Glossary. Issue 2.0, Document Number **WFMC-TC-1011.**

An experimental study of moment invariants and Fourier descriptors for shape based image retrieval

Atul Sajjanhar
Gippsland School of Computing and Information Technology
Monash University, Churchill
Victoria - 3842, Australia.
{atuls@fcit.monash.edu.au }

Guojun Lu
Gippsland School of Computing and Information Technology
Monash University, Churchill
Victoria - 3842, Australia.
{guojunl@fcit.monash.edu.au}

Jill Wright
Gippsland School of Applied Science
Monash University, Churchill
Victoria - 3842, Australia.
{jillw@gas.sci.monash.edu.au }

Abstract

Retrieval of images based on object shape is one of the most challenging aspects of content based image retrieval systems. In this paper we describe Fourier descriptors and moment invariants for shape based image retrieval and present results of an experimental study of the performance of the two techniques. The comparison between these two methods is done by indexing the shapes in a database for both the methods and making the same queries for both the methods. It is found that both the methods are comparable.

Keywords shape representation, image retrieval, pattern recognition, moment invariants, Fourier descriptors

1. Introduction

Current technology allows us to generate, scan, transmit, store and manipulate large numbers of digital images. However, current practice in accessing and retrieving images is still primitive. Today we access image databases based on captions. Although useful there are several problems with this approach, such as the fact that often the original keywords do not allow for unanticipated search in subsequent applications, and more important, inadequacy of uniform textual descriptions of such categories as colour, texture and shape. With the future availability of large image and multimedia databases, there will be a need to access images, using their content (shape, colour and texture), in addition to text queries.

To create a system that can answer queries based on image content we need to find which image features are significant and how these features can be used to search for the images of interest. The basic problem is that images differ from text format of data and hence image data requires a totally different technique of indexing and query processing. Queries about images require a visual language and new search and access mechanisms.

Much research is being done to develop tools for analysing images based on their content and then representing them in a manner that the images can then be searched based on these representations. An example is the **QBIC** project (Query By Image Content) [2, 4] in which the queries are based on colour, texture and shape. Content-based image retrieval systems can be represented as shown in Figure 1.

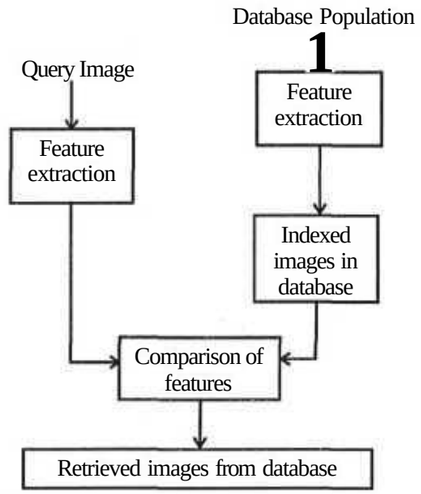


Figure 1 : Block diagram for content based image retrieval

Features are extracted from the images during population of the image database. The features extracted are based on low level features like shape of objects in the image, image colour and image texture. These features are used to index the images. At query time, the user enters a query into the system and features are extracted from the query image. The query image is compared with the database images by the image retrieval system on the basis of the closeness of the extracted features and the best match images are presented to the user.

Proceedings of the Second Australian Document Computing Symposium, Melbourne, Australia, April 5, 1997.

Shape based image retrieval is one of the content based image retrieval techniques. We present an introduction to shape based image retrieval in section 2.

2. Shape based image retrieval

In shape based image retrieval the shape of objects in images is used for the representation of images.

Segmentation is the first step of feature extraction in shape based image retrieval. Segmentation is the process to obtain the boundary of the constituent objects in an image. Segmentation of images requires the detection of discontinuities in an image which is based on detecting points, lines and edges in an image. One of the underlying theories for detecting discontinuities in images is based on the use of spatial masks [1].

After segmentation of an image into its constituent objects the boundary shape of the segmented objects is used for deriving an index for their representation. When deriving an index for the shape representation of an object the following requirements should be met in order to achieve a good similarity measure during image retrieval.

1. The representation of a shape should be invariant to translation, rotation and scale.

2. Similar shapes should have similar representations and dissimilar shapes should have different representations.

User queries are based on user constructed shape boundaries which can either be created directly or outlined from real images in the image database.

Many techniques have been used by researchers for shape representation and similarity measure. Some of the earlier techniques are summarised by Pavlidis [18]. A method based on turning angle which uses the turning of tangents drawn at points sampled along the shape boundary for shape representation was used by McConnell [15]. The class of curve fitting approach include the chain code approach used by Freeman [14] and McKee [23] and polygonal approximation used by Pavlidis [24]. A technique based on autoregressive model was first proposed by Kashyap **[19]. A** technique based on Hausdorff distance was used by Huttenlocher [17] which was modified and used by Scassellati [16]. Another category is the decomposition technique including medial axis transform (MAT) [20], decomposition at concave vertices [21], and decomposition by clustering [22].

From literature survey it has been found that the moments technique [5, 6, 7, 12, 13] and the Fourier descriptors technique [3, 8, 9, 10, 11] are the most popular techniques for shape based image retrieval. There is no literature on the comparison of these two techniques for shape based image retrieval. In this paper we describe and compare the performance of these two techniques experimentally. We perform the comparison by deriving feature vectors for the images in an image database for both the methods and then making the same queries for both the methods.

The moments technique and the Fourier descriptors technique for shape based image retrieval are described in section 3 and 4 respectively. We detail our experiment and present the results of a comparison of the performance between the moments technique and the Fourier descriptors technique in section 5. We present the conclusion in section 6.

3. Moments technique for shape based image retrieval

3.1. Definitions

Moments technique for shape representation and similarity measure has been extensively used in shape recognition. The moments of a shape boundary are uniquely determined by the shape boundary and are therefore used for shape representation and similarity measure. For a **2-dimensional** discretised image plane the moments of order p+q are given by,

$$m_{pq} = \sum_x \sum_y x^p y^q f(x,y) \qquad (1)$$

for p, q = 0, 1,2

where x, y are the pixel coordinates and f(x, y) are the pixel values which can take values of ones and zeros. There are two ways to calculate moments: boundary based and silhouette based depending on the values taken by f(x, y) which are as follows.

1. Boundary based moments : In this case f(x, y) in Eqn. 1 will take values 1 on the shape boundary and values zeros inside of and outside of the shape boundary.

2. Silhouette based moments : In this case f(x, y) in Eqn. 1 will take values 1 on and inside of the shape boundary and values zeros outside of the shape boundary.

The central moments of a binary image are given by,

$$\mu_{pq} = \sum_{x} \sum_{y} (\bar{x} - x)^p (\bar{y} - y)^q f(x, y) \quad (2)$$

where

$$\bar{x} = \frac{m_{10}}{moo} \qquad \bar{y} = \frac{m_{01}}{m_{00}} \quad (3)$$

3.2. Feature vector construction

We need to decide whether to use the moments computed from the image boundary or the solid silhouette. It has been shown that the moments computed from the image boundary contain more information about the high frequency portions of the image than those computed from the solid silhouette [6]. Therefore the minute details of the image are better characterised by the moments computed from the shape boundary. On the other hand, the general shape features of the image are better characterised by the moments computed from the solid silhouette. Since we are interested in the general shape features of images we decide to use the moments computed from the solid silhouette.

Normalisation is required in order to obtain feature vectors which are invariant to rotation, translation and scale. Moment invariants described by Hu [5] have been shown to be invariant to scale, translation and rotation. We use the moment invariants to derive the coordinates of the feature vectors for shape boundaries. Moment invariants are obtained from scale invariant moments for solid silhouette or shape boundary. The silhouette based moments normalised for scale are as follows.

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \quad (4)$$

where

$$\gamma = \frac{p + q}{2} + 1 \quad (5)$$

The moments obtained from Eqn. 4 are normalised for scale. These scale invariant, silhouette based moments are used to derive the moment invariants.

It has been shown that similar moment invariants do not guarantee similar shapes but using several moment invariants to describe shapes can overcome this problem since it is unlikely that dissimilar shapes will have several similar moment invariants. We use the first four moment invariants to construct the feature vectors for the shapes. The moment invariants are derived from the second and the third order normalised moments (Eqn. 4) and are computed as follows.

$$\Phi_1 = \eta_{20} + \eta_{02} \quad (6)$$

$$\Phi_2 = (\eta_{20} - \eta_{02})^2 + 4(\eta_{11})^2 \quad (7)$$

$$\Phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \quad (8)$$

$$\Phi_4 = (\eta_{30} + \eta_{12})^2 + (T|21 + \eta_{03})^2 \quad (9)$$

We use the logarithmic values of the moment invariants as the coordinates of the feature vector for shapes. The feature vectors thus obtained are of the form,

$$x = [\log(\Phi_1)\ \log(\Phi_2)\ \log(\Phi_3)\ \log(\Phi_4)] \quad (10)$$

The computation of similarity measure for the shapes indexed using the feature vectors of the form shown in Eqn. 10 is described in the following section.

3.3. Similarity measure

The Euclidean distance between the feature vectors of the query shape and the shapes in the image database is used as a measure of the distance between the query shapes and the shapes in the database.

The Euclidean distance between the feature vectors for 2 shapes A and B is computed as,

$$\text{Distance} = [\{\log(\Phi_{1A}) - \log(\Phi_{1B})\}^2 + \{\log(\Phi_{2A}) - \log(\Phi_{2B})\}^2 + \{\log(\Phi_{3A}) - \log(\Phi_{3B})\}^2 + \{\log(\Phi_{4A}) - \log(\Phi_{4B})\}^2]^{0.5} \quad (11)$$

where $\log(\Phi_{iA})$ for $i=1$ to 4 are the coordinates of the feature vector for shape A and $\log(\Phi_{jB})$ for $j=1$ to 4 are the coordinates of the feature vector for shape B.

A shape in the database is considered most similar to a query shape when the Euclidean distance between its feature vector and the feature vector of the query shape is minimal in the set of Euclidean distances computed between the query shape and all the shapes in the database .

4. Fourier descriptors for shape based image retrieval

Fourier descriptors have been used for shape representation and similarity measure. In the Fourier descriptors method the shape boundary is represented in terms of a function which is called the shape signature. Fourier descriptors for shape boundaries are derived from the Fourier transformation of the function used as the shape signature.

The discrete Fourier transformation of a shape signature f(i) is given by,

$$F_u = 1/N \sum_{i=0}^{N-1} f(i) \exp[-j27rui/N] \quad (12)$$

for u = 0 to N-1

where i - 0 to N-l are the N sample points along the shape boundary for which f(i) is computed. The computational complexity of the direct implementation of Eqn. 12 can be reduced by using fast Fourier transform. The coefficients obtained by the discrete Fourier transformation (DFT) (Eqn. 12) of the shape signature represent the shape in the frequency domain and are used for feature vector construction.

The functions which are commonly used as shape signatures are curvature function, centroidal distance function and the complex coordinate function. They are described in the following section.

4.1. Shape signatures

Using the curvature of the shape boundary as the shape signature requires the computation of the curvature at the sample points along the boundary which is calculated as the difference in two successive tangent values at the boundary. The curvature is calculated at the boundary pixel (x_i, y_i) by finding the angle between the 2 tangents: one drawn from (x_i, y_i) and the second drawn from the preceding boundary pixel at (x_{i-1}, y_{i-1}) to points separated by w pixels in the counter-clockwise direction (where w is the window size). This is illustrated in Figure 2.

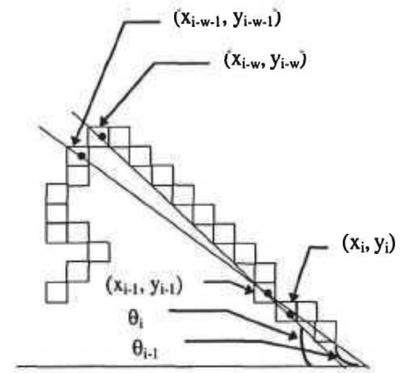


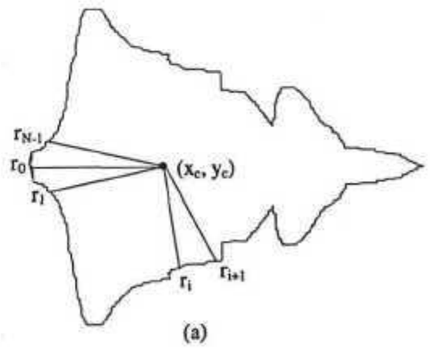
Figure 2 : Calculation of the curvature of a shape boundary

The curvature function for the boundary tangents is computed as,

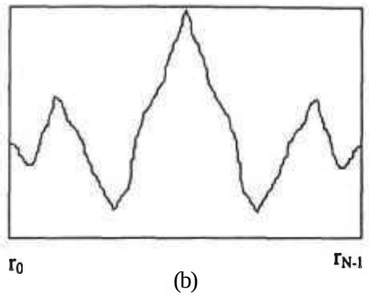
$$c_i = \tan^{-1}\left[\frac{y_i - y_{i-w}}{x_i - x_{i-w}}\right] - \tan^{-1}\left[\frac{y_{i-1} - y_{i-1-w}}{x_{i-1} - x_{i-1-w}}\right] \quad (13)$$

where c_i is the value of the curvature function at the sample point (x_i, y_i) on the shape boundary. The curvature calculated at the sample points along the shape boundary can be used as a shape signature.

The centroidal distances are also used as shape signatures. Centroidal distances are calculated at the sample points along the shape boundary by finding the distances of the sample points from the centroid of the shape boundary, this is shown in Figure 3.



(a)



(b)

Figure 3 : (a) The centroidal distances at the sample points along the shape boundary
(b) the plot of the centroidal distance function

The centroidal distances at the sample points along the shape boundary are shown in Figure 3(b), they are computed as,

$$r_i = \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2} \qquad (14)$$

for i - 0 to N-l

where (x_c, y_c) is the centroid and (x_i, y_i) for i-0 to N-l are the N sample points along the shape boundary as shown in Figure 3(a).

Another commonly used shape signature is the complex coordinate function. The complex coordinate function simply expresses the sample points along the shape boundary in terms of complex coordinates. The complex coordinates are computed as,

$$Zi-(Xc-Xi) + j\,(y_c - y_i) \qquad (15)$$

for i = 0 to N-1

where (x_c, y_c) is the centroid and (x_i, y_i) for i-0 to N-l are the N sample points along the shape boundary.

4.2. Feature vector construction and similarity measure

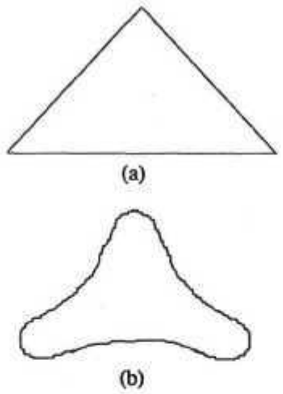
Feature vectors normalised for rotation, translation and scale are derived from Fourier coefficients obtained by the DFT of the shape **signature.** It has been shown that the performance of the Fourier descriptors technique is not significantly different for different shape signatures [3]. We use the centroidal distances as the shape signature owing to ease of implementation.

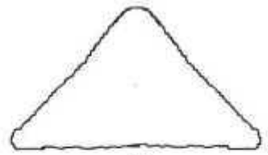
Rotation normalisation is achieved by taking the magnitude information only from the discrete Fourier transform of the function used as the shape signature. The phase information is ignored. Scale normalisation is achieved by dividing the absolute values of the descriptors by the absolute value of the DC component (which is F_0 in **Eqn.** 12). Translation invariance comes from the nature of the function used as the shape signature (Eqn. 14).

For obtaining the feature vector for the Fourier descriptors method we sample 64 points at equal distance along the perimeter of the shape boundary. We compute centroidal distances for the sample points (Eqn. 14) and use the centroidal distances as the shape signature. The discrete Fourier transformation of the centroidal distances of the sample points is calculated from Eqn. 12. The Fourier coefficients obtained are used to derive the feature vectors for the shape boundaries.

The lower frequency Fourier coefficients contain information about the general shape and the higher frequency coefficients contain information about the fine details. Since we are interested in indexing the shape boundaries for their general shape and not the fine details, we use only the low frequency coefficients for constructing the feature vectors for the shapes.

We need to decide how many Fourier coefficients carry sufficient shape information and then use them to construct the coordinates of the feature vector. Our experiments show that 8 coefficients are sufficient for most shapes. We show reconstructions of a shape with 4 and 8 Fourier coefficients in Figure 4.





(c)

Figure 4 : (a) original shape
(b) reconstructed shape with 4 coefficients
and (c) reconstructed shape with 8 coefficients

We use the 8 lowest frequency Fourier coefficients to construct the feature vectors for the shape boundaries. The feature vectors are mathematically represented as,

$$x = \left[\frac{|F_1|}{|F_0|} \cdots \frac{|F_7|}{|F_0|} \right] \qquad (16)$$

where F_0 to F_7 are the 8 Fourier coefficients obtained by the Fourier transformation of the centroidal distances of the sample points along the shape boundary. The feature vector in Eqn. 16 conforms with the requirement for invariance to rotation, scaling and translation.

During database population stage the feature vector is calculated for each shape added to the database. During querying the feature vector is calculated for the query shape and compared with the feature vectors for the shapes in the database. The difference between a query shape and the shapes in the image database is calculated as the Euclidean distance between the feature vectors.

5. Comparative study

We implemented an image retrieval system for indexing the shapes in a database for the moments technique and the Fourier descriptors technique and subsequent image retrieval based on similarity measure between the query shapes and the shapes in the database. This is described in the following sections.

5.1. User Interface

The user interface in our image retrieval system has two parts. One controlling the database population stage and the other for performing queries. The interface at the database population stage allows the user to enter shapes using the mouse interface. These shapes are then indexed and added to the database.

The interface for entering queries allows the user to draw shapes using the mouse (direct query). Alternatively, the user can also enter a shape already existing in the database (query by example) as a query. The system then retrieves shapes similar in shape to the user query from the database in decreasing order of similarity.

5.2. Experiment and results

The database used for the experiment contained 160 shapes. For each query shape the perceptually similar shapes in the database were identified. The retrieval system produced a sorted list of shapes in decreasing order of similarity to the query shape. The shapes in the database were separately indexed for the Fourier descriptors technique and the moments technique. Same queries were then performed for the Fourier descriptors technique and the moments technique. The seven queries used are as shown in Figure 5.

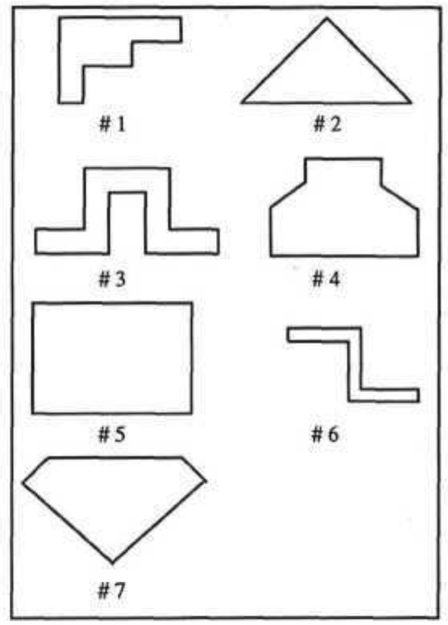


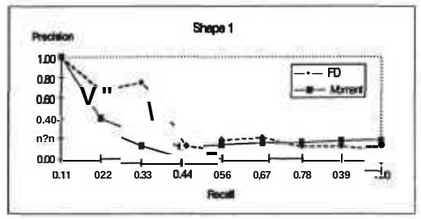
Figure 5 : Seven query shapes

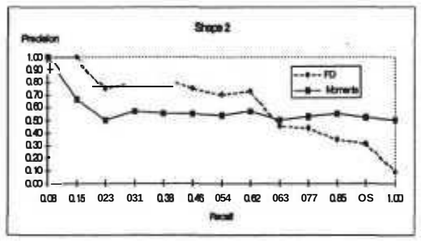
The performance of the retrieval techniques was measured in terms of the recall and the precision.

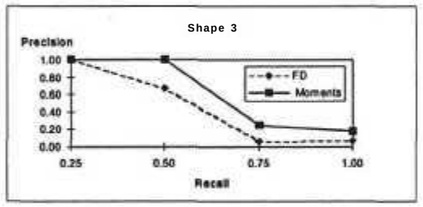
Recall measures the ability of retrieving relevant information items from the database. It is defined as the ratio between the number of relevant items retrieved and the total number of relevant items in the database. Precision measures the retrieval accuracy and is defined as the ratio between the number of relevant items retrieved and the total number of items retrieved.

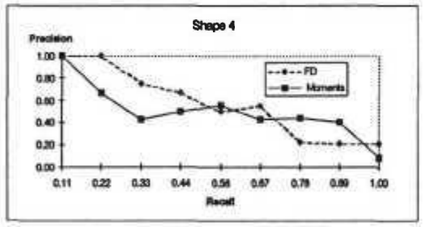
For each query object the relevant items in the database are the object shapes which are perceptually similar to the query object shape.

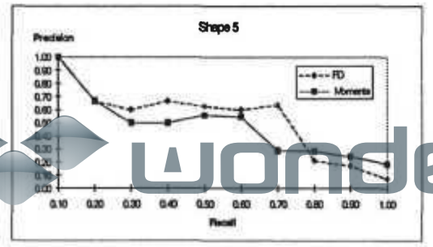
The values of recall and precision for the seven query shapes are plotted in Figure 6 for moment invariants and Fourier descriptors.

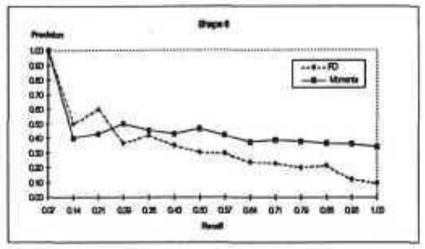












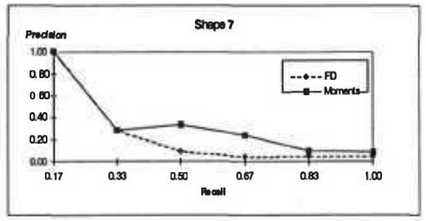


Figure 6 : Graphs showing the recall and precision for the 7 query shapes

The steps in which recall increments along the x-axis depends on the number of relevant shapes in the database for the particular query shape. For example, the number of relevant shapes for shape 1 are 9, so the recall increments in steps of $1/9 \approx 0.11$.

6. Conclusion

From the results above it appears that the Fourier descriptors method and the moment invariants method perform equally well. We averaged the results obtained for the seven query shapes for both the methods and obtained the results shown in Figure 7.

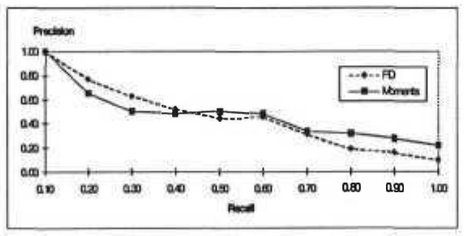


Figure 7 : Recall and precision averaged over seven queries for moment invariants and Fourier descriptors

From the graph in Figure 7 it is concluded that both the methods perform equally well and there is no clear winner.

The moments technique requires more computation than the Fourier descriptors technique since we use the silhouette based moments. The computation requirements for the moments technique can be made comparable with that for Fourier descriptors technique by using the boundary based moments.

References

1. R. C. Gonzalez and P. Wintz. *Digital Image Processing.* Addison-Wesley, Second Edition, 1987.

2. J. Ashley et al. Automatic and Semi-Automatic Methods for Image Annotation and Retrieval in QBIC. *Storage and Retrieval for Image and Video Databases III, SPIE Proceedings,* Vol. 2420, pages 24-34, 1995.

3. H. Kauppinen, T. Seppanen, M. Pietikainen. An Experimental Comparison of Autoregressive Fourier-Based Descriptors in 2D Shape Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence* , Vol 17, No. 2, pages 201-207, February 1995.

4. W. Niblack et al. The QBIC Project : Querying by Image Content using Color, Texture and Shape. *Research report, IBM Almaden Research Center,* San Jose, CA, 1993.

5. M. K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory,* Vol. IT-8, pages 179-187, February 1962.

6. A. S. Dudani, K. J. Breeding, R. B. McGhee. Aircraft Identification by Moment Invariants. *IEEE Transaction on Computers,* Vol. C-26, No. 1, pages 39-46, January 1977.

7. S. O. Belkasim, M. Shridhar and M. Ahmadi. Pattern recognition with moment invariants: a comparative study and new results. *Pattern Recognition,* Vol. 24, No. 12, pages 1117-1138, 1991.

8. E. Persoon and King-Sun Fu. Shape Discrimination Using Fourier Descriptors. *IEEE transactions on Systems, Man and Cybernetics,* Vol. SMC-7, No. 3, pages 170-179, March 1977.

9. C. T. Zahn and R. Z. Roskies. Fourier descriptions for plane closed curves. *IEEE transactions on Computers,* Vol. 21, No. 3, pages 269-281, 1972.

10. R. Chellappa and R. Bagdazian. Fourier coding of image boundaries. *IEEE transactions on Pattern Analysis and Machine Intelligence,* Vol. 6, pages 102-105, 1984.

11. A. Krzyak, S. Y. Leung and C. Y. Suen. Reconstruction of two-dimensional patterns from Fourier descriptors. *Machine Vision and Applications,* pages 123-140, 1989.

12. R. J. Prokop and A. P Reeves. A survey of moment-based techniques for unoccluded object representation and recognition. *Graphical Models and Image Processing,* Vol. 54, pages 438-460, 1992.

13. Y. S. Abu-Mostafa and D. Psaltis. Image Normalisation by Complex Moments. *IEEE transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-7, No. 1, pages 46-55, 1985.

14. H. Freeman. Computer Processing of Line-Drawing Images. *Computing Surveys,* Vol. 6, No. 1, pages 57-97, March 1974.

15. R. McConnell, R. Kwok et al. ψ - s correlation and dynamic time warping : Two methods for tracking ice floes in SAR images. *IEEE transactions on Geoscience and Remote Sensing,* Vol. 29, No. 6, pages 1004-1012, November 1991.

16. B. Scassellati, S. Alexopoulos and M. Flickner. Retrieving images by 2D shape: a comparison of computation methods with human perceptual judgements. *Storage and Retrieval for Image and Video Databases II, SPIE Proceedings,* Vol. 2185, pages 2-14, 1994.

17. D. Huttenlocher, W. Rucklidge and G. Klanderman. Comparing images using Hausdorff distance under translation. *IEEE* pages 654-656, June 1992.

18. T. Pavlidis. Algorithms for shape analysis of contour and waveforms. *IEEE transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-2, No. 4, pages 301-312, July 1980.

19. R. L. Kashyap and R. Chellappa. Stochastic models for closed boundary analysis: Representation and reconstruction. *IEEE transactions on Information Theory,* Vol. IT-27, No. 5, pages 627-637, Sept. 1981.

20. H. Blum. A transformation for extracting new descriptions of shape. *Proc. Symp. Models Perception Speech Visual Form., Cambridge,* MA: MIT Press 1964.

21. H. Y. Feng and T. Pavlidis. Decomposition of polygons into simpler components: Feature extraction for syntactic pattern recognition. *IEEE transactions on Computers*, Vol. C-24, pages 636-650, 1975.

22. L. G. Shapiro and R .M. Haralick. Decomposition of two-dimensional shapes by graph theoretic clustering. *IEEE transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, pages 10-20, 1979.

23. J. W. Mckee and J. K. Agarwal. Computer recognition of partial views of curved objects. *IEEE transactions on Computers*, Vol. C-26, No. 8, pages 790-800, 1977.

24. T. P. Pavlidis. Polygonal approximation by Newton's method. *IEEE transactions on Computers*, Vol. C-26, No. 8, pages 800-807, August 1977.

Reuse of Information through virtual documents

Anne-Marie Vercoustre [1]

INRIA — Rocquencourt,
BP 105, 78153
Le Chesnay Cedex, France

Anne-Marie. Vercoustre@inria.fr

Jon Dell'Oro, Brendan Hills

CSIRO Mathematical and Information Sciences
723 Swanston Street, Carlton,
Victoria, 3053, Australia

Jon.DellOro@cmis.csiro.au
Brendan.HiUs@cmis.csiro.au

Abstract

This paper explores the issue of representing textual information in the form of virtual documents that include data and fragments of documents from remote sources — especially from databases and SGML document databases. virtual documents are dynamically generated, and therefore always present up-to-date information when they are instantiated.. The benefit of this paradigm is that it allows information to be shared, reused, and adapted for various contexts. A Virtual document specification defines how to find and retrieve information objects from databases or from existing documents, and how to assemble it into another document. Virtual documents can be used to create HTML pages that contain information from one or several remote or local databases, to assembly parts of existing documents into a new one, or to define various views of the same information according to various needs. This paper focuses on the prototype implementation of virtual documents from the perspectives of document authoring and architecture. We propose an SGML syntax for Information Object that includes OQL-queries for retrieving fragments of existing documents, transformations on an intermediate tree representation, and output mapping to the virtual document structures.

Keywords Document Databases, Document Management, HTML, Hypermedia, Information Discovery, SGML, virtual documents.

1 Introduction

An enormous amount of government and corporate information now resides in electronic documents, and document management has replaced data management as a focus of computing research over the last two years. The World Wide Web has gained widespread popularity as a means to access this information, both for private use within an organisation as well as for public availability. The Internet and the Web provide the basic mechanisms for information discovery in widely distributed heterogeneous networks, but more advanced support is necessary for finding and providing the right information to the right people.

When we approach a very large collection of dynamic documents, such as the WWW or even private corporate nets (intranets), our main issues for publishing information are:

- automatic updating of the information contained within documents.'
- configurability of the documents according to a user's needs.
- sharing and reuse of information through various documents.
- automatically and flexibly providing meta-descriptions to deal with evolving required descriptions.

Publishing information on large networks is still mostly done by writing the information in static HTML pages, making it difficult to update, share, process, and reuse information. Another practice is to dynamically generate HTML pages by running scripts or programs on the web server. When the server is a gateway to a database the displayed information reflects the current state of the database and is always up-to-date. However, in this approach, there is no easy way of building other virtual pages using the same information from the database or information from several databases.

The Internet and the Web provide the basic mechanisms for Information Discovery in widely distributed heterogeneous networks, but, in spite of the increasing number of good Web search engines and sophisticated query interfaces, more advanced support is necessary for finding the right information for the right people, and presenting it in the appropriate form.

A new trend is to build applications which are able to directly extract information from Web pages

' Most of this author's work performed while working al CSIRO Mathematical and Information Sciences, Melbourne, Australia.

Proceedings **of the Second Australian Document Computing Symposium, Melbourne Australia, April 5 1997.**

and to reuse it to provide the user with new information more suitable for their specific need or task. For example, TransactNet has developed a tool called Wit [24] that, when given any Web page, can build and compile Java classes corresponding to the "objects" in the page. Each time this page or any page of the same model is given to Wit, Wit will construct the Java objects corresponding to the page. The objects can then be reused from other Java applications. Although WIT uses a limited and low level model of pages — built from analysing HTML tags and not supporting lists or recursive structures — it can be very useful for reusing specific Web information, for example by regularly issuing the same query to a server to check if there is something new, and only then notify the user.

Araneaus et al [1] also reuse a set of regular Web pages by building a page oriented data model to describe the schema of a structure in the spirit of database schema. It is possible to define a relational view on the server that combines path queries for extracting information according to the model, and follows links to other pages to get other information. This view can in turn be transformed into new Web pages with a different structure.

These two examples are based on the idea of reusing information from regular Web pages, i.e. pages that have been produced automatically from a database, or generated from a script. In any case the model of the pages has to be "guessed" from the HTML output. It must also be general enough to apply to a variety of Web servers, and must be recreated each time the server changes its output format, or new HTML features appear.

Our approach is to reuse information from SGML documents to take advantage of the rich structure model provided by the DTD (Document Type Definition), which guarantees that the document will conform to the model.

This paper examines the issue of reusing information as virtual documents that include data and fragments of documents from remote sources, especially from conventional databases and SGML document databases. Virtual documents can be designed on a different server than the one where the initial sources of information are located, and can be instantiated on the client side. It is an extension to the current practice of dynamically generated documents, an example of which is a CGI script for the WWW.

This paper also gives the requirements of a design language for building virtual documents. A virtual document description will include SGML-based specifications of the logical structures with **OQL-like** queries for retrieving content parts from a database (See Cattell [6] for a description of OQL). The virtual document description will therefore provide a constructive specification of the document.

2 Reuse of information objects

Levy [9] has identified reuse as being central to the creation and use of documents. To a great extent, the work of producing new documents involves reusing pieces of previously existing documents — where reuse involves finding relevant material, modifying it as needed, and assembling the pieces together.

The objective of our current project is to build a system for reusing SGML documents and other information sources in order to:

- create new documents from existing ones and from various sources of information
- provide customised and task-oriented views upon existing documents. (As described by Beir and Goodisman [3])
- extract information objects from SGML documents and make them accessible to distributed external applications

This approach regards a "document" as a set of information objects that can be retrieved and combined in different ways and reused in various contexts, as described by Skar [14] in his discussion of processing SGML documents as collections of small parts, rather than as monoliths.

This paper focuses on the first objective of the list above, i.e., the creation of documents in a flexible way from distributed sources of primary information.

2.1 Background

For many web publishers today, information is published in two common ways: either stored as static HTML pages, or as web pages that are dynamically produced by running scripts or programs on a web server. In the first case, the information is the HTML document itself, so that it is difficult to update, share, process, and reuse.

Hypertext and the WWW have long been regarded as good ways of sharing and reusing information, since hyperlinks allow references to existing pages without duplication. This approach encourages the browsing paradigm of accessing information, which may be time consuming and prevent the reader from gaining an synopsis of several pieces of a document. Some earlier hypertext systems, such as Guide [4] offered an included mechanism for expanding links within the current page, but this required an accurate taxonomy of the set of hypertext documents, which obviously the World Wide Web cannot support.

Moreover, Web pages are rarely short and concise, and they seldom include useful structural information. For these reasons, Web pages do not offer the appropriate granularity for inclusion into other documents, as they may contain information content on a range of topics, intermixed with presentation elements that would be undesirable in another context.

HTML was initially designed for describing the organisation of the information and its semantic content, but in current day use, it is used as a presentation-oriented language. As such, HTML is inappropriate for identifying and accessing specific parts of information for a particular purpose.

It appears that information management approaches have been moving from a very rigid database approach, dealing with very structured data, to a world of flat text which is supposed to represent all kinds of information. This situation is patently unworkable. Using more structured document formats (e.g. SGML) can benefit a lot from database support, especially when documents are regarded as sets of information objects.

The publication of information on the Web should be the dynamic creation of documents from various sources of information which can be assembled in various ways. The next section will define virtual documents and review the current approaches in creating dynamic documents for the Web.

3 Dynamic and virtual documents

3.1 What is a virtual document?

While there are millions of static pages on the Web, there are an increasing amount of dynamically generated pages. The simplest case of a dynamically generated page is an HTML document that is constructed in response to a search query. All search engines on the Web return pages such as these, displaying a list of answers in the form of links to the actual documents and some information about the content (Title, abstract or extract of sentences etc.). Although such documents have a persistent name[2], and the content is virtual (since it can change over time), it is questionable whether a document like this can be called a virtual document or not

Virtual documents have been defined by Gruber [8] as hypermedia documents that are *generated on demand*, in response to user input. This general definition seems at first to apply to the previous example, until we consider whether those pages are actually "documents" or whether they just look like documents. The question "what is a document?" [20] is a difficult one, but a first definition could be that a document has two main functions: to communicate information, and to assemble and organise this information to make it clear and usable to the reader.

Those pages generated by search engines only assemble information which may or may be not relevant to the user, may belong to quite different domains, or may appear in several places. We are interested in the authoring and construction of virtual documents which are intended to *assemble and organise information* for supporting specific tasks or decision process.

Rather than the previous definition, we would define virtual documents to be hypermedia documents as dynamic documents whose structure and content can be easily adapted to the specific user. They can therefore be regarded as a flexible means to reuse information in a variety of contexts.

[2] Here's an example of a page which has a persistent name, yet the content is dynamic:
`http://altavista.digital.com/ CGI-bin/query?pg=q&what=web&fmt=.&q=Virtual+and+Documents`

3.2 Some examples of virtual documents

Virtual documents are well-suited to present up-to-date information in domains where information is changing frequently, or to adapt themselves to the reader's communication needs. The WWW offers many good examples of virtual documents:

- Virtual conference lists [16] that displays conferences events sorted by months, call for paper, date of events, or country.
- Interfaces to databases for which a WWW gateway has been developed. They offer two different functions: database querying and updating through form-based interfaces, or database browsing by generating pages with embedded links. An example of the latter is the 02 Auto demonstration [21].
- **Peba-II** [23]is a natural language generation system which dynamically produces a hypertext description or comparison of animals based on a knowledge base of animals and a discourse and text planning component.
- DME [8]is a **Question-Answer** system that generates domain based-explanations with embedded links for follow-up questions.

These virtual documents are actually rather advanced examples, especially the two **last** ones which involve natural language generation. Peba II, DME, and the 02 Web browser all use a domain model for generating pages with embedded links. The text itself is generated by the application using techniques of varying degrees of sophistication, ranging from specific or generic programs which use hierarchical object classes, to natural language generation using text planning.

An instance of these virtual documents has a persistent context independent identifier (a unique URL) that allows a user to reference or reuse the document in another context. In fact the URL is simply the query that generates this instance of the virtual document.

The generation of the above virtual documents, however sophisticated, is completely determined by the application. The end user or another application accessing the same domain database cannot reshape it

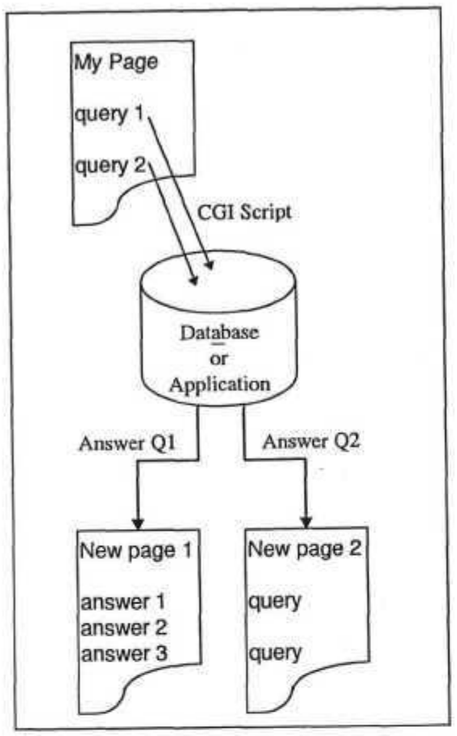


Figure 1: Server Driven Architecture

for another purpose. In this paper, we are mostly interested in describing virtual documents that allow the reuse of fragments of information which are distributed on remote servers or databases.

3.3 Architecture and processing

Typical applications which generate dynamic documents, such as those given above, issue calls to a Common Gateway Interface (CGI) program with specific parameters from an HTML document (either using a form or directly encoded in a URL). The HTTP server invokes the CGI program (possibly through a gateway to the application). The result of the CGI program is usually an HTML document built by the application that is passed to the client through the HTTP server. This model is shown in Figure 1.

In Figure 1, the virtual documents *New Page 1* and *New Page 2* are generated by the application and contain embedded links which may generate further call to CGI scripts if they are activated. In this model, called querying by browsing, the user is given more information by browsing through pages that are generated only if the user follow a particular path.

However, accessing the desired information can be very time consuming, and it can be difficult to get an overview of a sophisticated set of pages. The structure of the resultant pages are completely determined by the web programmer. We want to take the opposite approach, i.e. a "document driven" approach, in which a virtual document combines various information chunks instantiated by queries to external applications or a databases. This approach is shown in Figure 2.

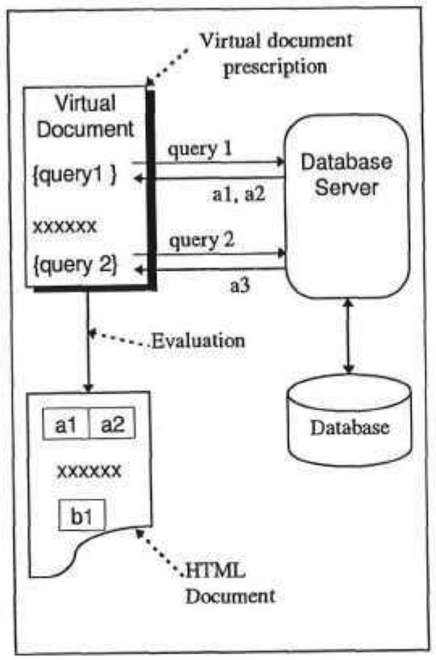


Figure 2: Document Driven Architecture

In Figure 2, the virtual document "prescription" gives any static parts of the virtual document, along with instructions about how to get the other information objects which make up the virtual document. All of the queries within the virtual document prescription are instantiated at the same time and their results are included in the document according to the instructions in the virtual document prescription. The organisation of the information within the virtual documents can be then driven by the usage of the information, rather than by the server's internal model of this information. It means also that information can be retrieved from various databases and documents, and reconfigured independent of the database developers and document authors.

3.4 Virtual documents on the Web

The model described in Figure 2 requires as a minimum a mechanism for including new elements into a document. Surprisingly, HTML does not offer this capability. More specific techniques for including pieces of information within a static HTML page include:

- an HTML document that contains the <IMG> tag with an URL for selecting the image to be included in the document. The URL can specify a file name or a request to a database or a CGI script that return information (e.g. the value of a counter). The WWW Consortium (W3C) is currently working on an HTML extension for multimedia objects insertion within a document through a new tag called `<OBJECT>` as a generalisation of the `<IMG>` tag [25].
- an HTML document that contains parts to be included at loading time using the include facility on the server side.
- "frames" are a Netscape extension to HTML that allows for displaying various sources of information in the same browsed page. Frames do not provide inclusion within the same HTML document; instead, they facilitate inclusion of several HTML pages within the same browser page in a bounded-box layout scheme.
- Java applets for inserting executable code.

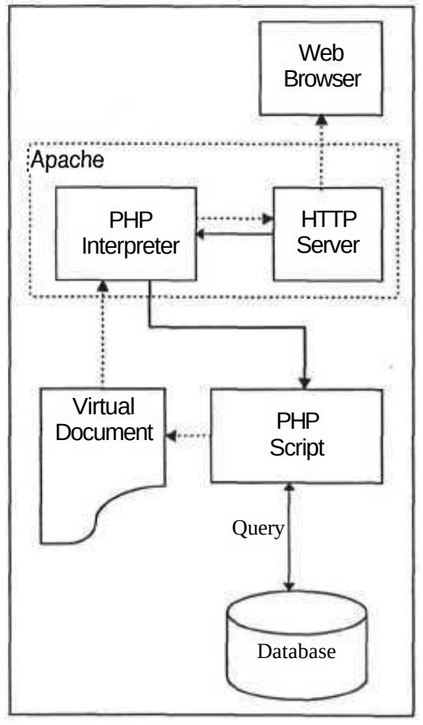


Figure 3: PHP Architecture

- advanced interfaces to databases — for example PHP as shown in Figure 3.

3.5 Virtual documents with SQL

We have worked with several examples of virtual documents to gain experience in various techniques for implementing them, both at the authoring level and at the architecture level. We will report here on a conference list with its content built from a database is a list of conferences about Hypertext, the Web, Electronic Documents, Advanced databases, Knowledge bases, etc. — all subjects of interest for our research group. The conference lists aims to be user-friendly, displaying the lists with various sorting criteria, such as topics, countries, dates or calls for papers. Additionally, extra conferences may be added by anyone on the web using a form.[3]

The data for this conference list is stored in an SQL database called mSQL. This is a UNIX-based database that implements a useful subset of ANSI SQL. We access the database using a web based interpreted language called PHP/FI [18]. This is a rich, mature wrapper for web pages that includes access control, support for database connections, language statements, and string processing. Among the many useful features of PHP is the ability to function as a module to the UNIX web-server Apache (as shown in Figure 3), thereby running in the same process space and performing operations very quickly.

The PHP system works by embedding SQL queries and control instructions in HTML comments, which is similar to other systems such as w3-msql [19]. Our system contains only two HTML documents (actually HTML files with embedded scripting, to be processed by the PHP interpreter). The first document defines the various conference lists while the second one is a form for adding or modifying a conference entry. The first document is very complex since it controls:

- content of the document by issuing the appropriate queries according to the selected sorting criteria.
- organisation of the document by creating the appropriate grouping of information.
- presentation of the document by embedding the content into HTML tags.
- the user interface for switching from one list to one other and from the browsing mode to the editing mode. It uses state information for adapting the interface and the display according to the context. For example, the page background colour is changed to "bisque" when in editing mode to remind the user to be careful in editing the database.

[3] .The URL for the conference list is `http://www.mel.dit.csiro.au/~vercous/Conference-list.html`

A very small part of the Conference list virtual page is shown in Figure 4.

```
<HTML>
 <Hl>Conference list for
Text Information Management</H1>
 sorted by:
<! — open the database —>
  <?
    putenv("MSQL_TCP_PORT=1112");
    msql_connect("localhost");
    $database="confs";
  >
  <?
    $result= msql($database,
      "select Title,Location,URL
      from TIM where b_year >= 1996
      ORDER BY year,b_month,b_day)";
      $num=msql_numrows($result);
      $i=0;
   >
<hr>
  <? while ($i<$num);
<b>
  <?
    echo msql_result
       ($result, $i, "Title");
  >
</b>
<br> <a href=
  <?
    echo msql_result
      ($result, Si, "URL");
  >
 >
  <?
    echo msql_result
      ($result, $i, "URL");
  >
 </a>
<hr>
  <?
    Si++;
    endwhile;
  >
</BODY>
</HTML>
```

Figure 4: Conference List Example

It is perhaps not surprising that the conference list which started as a very simple virtual document actually looks like a relatively complex program. To some extent, this is no better than a **CGI-script.**

From the architectural point of view, our virtual documents are actually scripts running on the PHP interpreter and accessing data from a database. The PHP interpreter handles a straight connection with the database (Figure 3).

This is a two-tiered architecture similar to the one described by Duan [7] for database applications on the Web using Java. Unlike JavaScript, **PHP/FI** is not browser dependent as JavaScript is a client-side HTML-embedded language, whilst **PHP/FI** is a server-side language.

The main feature is that the virtual documents "constructors" and the data are not on the same server and can be designed by quite different people. From the design point of view, this feature is very important, because the information reuse becomes feasible. Anyone using a PHP server could design easily their own conference list from our conference database (providing we give reading access).

From the authoring point of view, using PHP was at first satisfactory when the document were ordinary HTML documents with queries embedded into comments. It seemed so much more convenient than managing HTML and CGI files separately. Also, this enforces the document-oriented design of the virtual document, which was exactly what we were looking for. Perhaps unfortunately, we soon began to appreciate the utility and convenience of the control capabilities of the embedded programming language so our virtual document ended up looking like a program with some print statements for writing HTML tags: rather more like a CGI-script than an HTML document. Although we found this undesirable, it was perhaps inevitable because the scripting language is not declarative, but procedural.

Despite the breakdown of **"declarativeness",** the scripting approach still has some advantages:

- It is very easy to construct simple documents and the result still looks Uke HTML.
- Complex operations can benefit by putting everything in the same file, with many control variables, to produce many different pages. The resulting pages share whatever it is possible to share and there is **only** one page to maintain.
- Integration with various databases is tackled by the PHP server, and you only need to know the query language syntax and your database schema.

4 Virtual documents from SGML databases

We are interested in virtual documents that support the use, reuse and collaborative authoring of documents and especially documents that can combine information from distributed databases and fragments of existing SGML documents.

As an example of this sort of documentation system, consider writing manuals for personal computers. Each model of personal computer contains a number of *components* such as hard drives, sound cards and monitors, and each component has a number of *parts* such as power supply, fan or SCSI connector. The aim of the documentation effort is to produce a complete manual for each model of computer, and for those manuals to be automatically updated when the details of a lower level part change. This would be accomplished by having a query embedded in our personal computer manual which returns the required sections of the manual for each particular component. The component manuals would then in turn have queries which return the appropriate manual pages for each of their parts.

4.1 Document granularity and storage

The storage of SGML documents in databases has been the object of much research and development to improve various aspects of document management. In particular, various query languages and systems have been proposed to combine searching for documents base on their content as well as using queries based on document structure.

The way that documents are stored in the database has a large impact on the granularity of fragments of documents that can be retrieved. The granularity for searching can be different of the granularity for retrieval. For example SIM, the "Structured Information Manager", developed by **RMIT,** Australia [17], stores the full SGML text while indexing documents according to part of the structure as defined by the DTD.

Christophides et al [5] store SGML documents into an object database, namely the 02 object database system [22], using a database schema that maps onto the DTD. This allows for very powerful queries on the structure of the documents and the retrieval of any fragment of document, with some extra cost in storage and current performance as search using textual predicates is not integrated in the query optimiser.

Therefore there are two levels of queries:

- queries to get a set of objects from the database by issuing a query to the database or by giving its explicit address (a URL, or a query to retrieve a specific object in SIM, 02, or the Web)
- a query within an object using the "object model": for example getting an attribute of an object, or a section of an SGML document.

To perform the latter sort of query on heterogeneous objects we need some sort of intermediary format (e.g. a generic tree). The construction of this tree will be part of the interface to the database. For example for SGML documents stored as SGML files, the tree will be built by using the SGML parser corresponding output to the DTD. It would even be possible not to build the tree corresponding to the full document, but only the parts corresponding to the queried objects, using an approach described in Abiteboul et al. [2].

4.2 Requirements for authoring virtual documents

The main requirements for a virtual document language are:

- Parts of the document which are static should be as easy to edit as usual documents. The document should not be generated by a program.
- All the queries are evaluated at the same time (when loading the document).
- Answers to queries can be reorganised within the document. For example, a set of answers can be matched to a table or a list after being sorted.
- The answer to a query can be included in the form of the actual data or a link to the data, which accounts for a level of materialisation of the query.
- Virtual documents can be defined recursively, i.e. virtual document can contain queries or links to other virtual documents.
- The virtual document specification should contain control instructions for adapting the document to the actual retrieved data (e.g. when there is no answer to the query) or to an user profile.
- The virtual document can include parameters by metavariables representing queries to be instantiated.
- The database should be updatable from a virtual document instance by using an appropriate editor (though this is a difficult problem).

When the document is loaded, the virtual document is "evaluated" (using functional language terminology) and then displayed. The evaluation may be the generation of an **SGML/HTML** document that will be displayed using an SGML/HTML browser.

Our short examples above fulfil the requirement to access distributed sources of information, but not the requirement for suitable design support. We can consider two approaches to improve it: first, to extend a language for document description like SGML or HTML; second, to extend a document editor for supporting the retrieval, control and inclusion of external information within the document. These two approaches are complementary since the editor will need a language description for saving the virtual document. We will consider only the first approach here.

4.3 Specification language

Our experience with PHP has resulted in a better understanding of the various components needed for the description language. The language wiU include:

- query language for selecting and controlling the content elements.
- mapping language for transforming the previous elements into document elements.
- construction language with control capability for building the logical structure of the document; this language would include for example constructors for elements, lists, or fixed structures.
- presentation **language.**
  (see Figure 5)

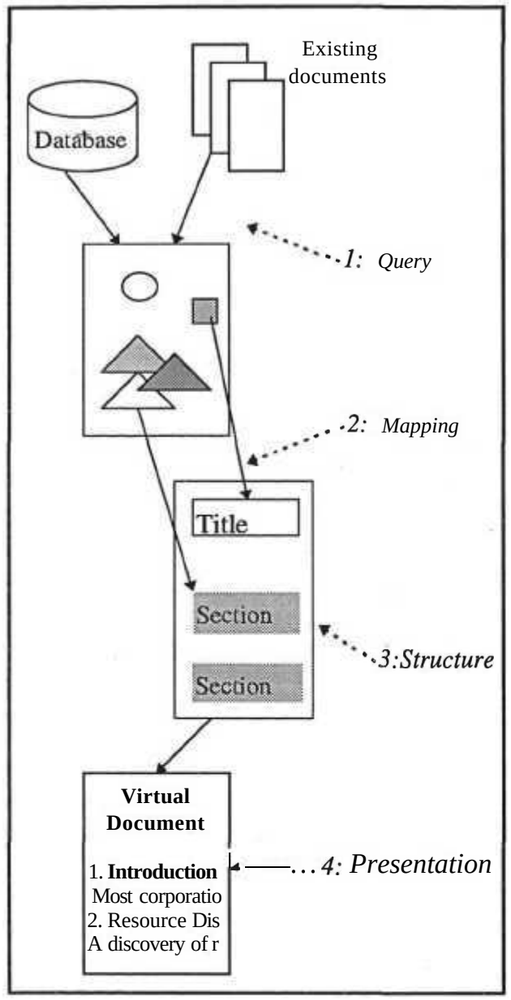


Figure 5: **Document** construction

The language interpretation process can be described as following:

1. The query language can specify which pieces of information have to be included into the document. It can be data from a database or fragments of existing documents.
2. The mapping language specifies the transformation of the information objects into document objects. For example, if the virtual document is an SGML document that includes fragments of other SGML documents, a mapping must be defined between SGML document elements to be included and SGML elements in the virtual documents, according to their DTD. This mapping could be defined through a middleware data model using labeled trees as proposed by Abiteboul et al. [2].
3. The construction of the logical structure of the document must include *operators* for building complex document elements like lists or aggregates. It must also include control capabilities for adapting the document structure to the environment (state variables or environment variables) and the actual content. Also, the construction of the logical structure of the document defines the static part of the document. The major difference of this to an SGML DTD is that the virtual document is a *constructive* description of the document and not a generic description for a class of document. However, the description can take advantage of a DTD definition by not redefining all the intermediary structures or common elements.
4. The presentation description specifies how to format and display the document. It also includes control instruction. If the virtual document is a HTML document, this part will not be distinct from the structure definition, since both would be determined by the actual tags. In the case of SGML documents, the presentation can be generic to a DTD and specified for each SGML editor/browser in use.

The Information objects will be referenced from a virtual document prescription using the **"InformationObject"** tag which is designed to work in exactly the same way as any other SGML tag. An example of the InformationObject tag is shown in Figure 6.

Note that in Figure 6, the content of the `<Output>` element is an SGML structure which conforms to the virtual document's DTD at the point of the virtual document prescription which is occupied by the `<InformationObject>` element. This structure contains variable names at the points where the transformation process will insert data from the queried information object. This approach has been used so that the virtual document system can easily provide a "dummy" information object which means that the virtual document can be validated as correct SGML. To do this, the system simply has to remove the InformationObject tag to the Output tag, and then the Output and InformationObject close tags. This will leave the contents of the Output tag which is valid SGML.

```
<DoO
<p>blah blah blah blah.
<InformationObject>
  <Query Method="SQL",
    Database= "Parts">
      select blabla
  </Query>
  <Transformation>
    <Process>
        (processing          instructions...)
    </Process>
    <Output>
      <NumberedList>
        <ListItem>
          ${Varl)
        </ListItem>*
      </NumberedList>
    </Output>
  </Transformation>
</InformationObject>
<p>blah blah blah.
</doc>
```

Figure 6: `<InformationObject>` Tag Example

We think that the International Standard for Document Semantic and Specification Language, DSSSL [15], could be a good candidate for defining the transformation part of virtual documents. Indeed DSSSL defines the syntax, semantic, and processing model of two languages for the specification of SGML documents processing, described here:

1. The transformation language for transforming SGML documents **marked-up** in accordance with one or more DTD's into other SGML documents marked-up with another DTD. It provides access to, and control of, all possible marked-up information in an document through the Query Language component of DSSSL.
2. The style language that define the presentation of a document.

DSSSL includes an expression language to create and manipulate objects. The DSSSL expression language is based on the Schema Programming language (a functional language like Lisp). It may not be surprising that the previous example uses a Lisp-like syntax, as we found it convenient for defining such mapping and constructions.

4.4 Related work

MacWeb[4][11] synthesises documents by dynamically combining fragments of information which are organised in an object-oriented knowledge base. The links to scripts which trigger generation of the document are placed in the WebOfTypes. The synthesis of the document is done according to a task model. MacWeb uses a script language (WebTalk) for describing the logical structure and the semantic content of the virtual document. The interpretation of the script generates the virtual document.

The main difference with our approach is that MacWeb uses an object oriented knowledge base (semantic representation of the content) while we use more standard representations of information like SGML document databases or general databases. A more important difference is that MacWeb generates the layout of the document from methods attached to the Objects types, making it impossible for various external applications to redesign them, as pointed out in section 3.3.

[4]This Hypertext prototype should not be confused with the Macintosh Web Browser of same name.

5 Conclusions and further work

Virtual documents offer an effective way of publishing and flexibly sharing information, independent of the server or database on which the information is located. It solves the problem of updating various documents that share the same information which will be updated only once. It also allows users to reassemble information according to various tasks, new objectives, or different contexts, states or environments. PHP has proven to be a very convenient scripting language for building HTML virtual documents that include data from an SQL database. A subpart of the DSSSL language could be a good candidate for defining more complex SGML virtual documents.

6 Acknowledgements

This work has been partially supported by the Cooperative Research Centres Program through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia.

References

[1] P. Atzeni, G. Mecca, P. Merialdo, E. Tabet. Structure in the Web. Submitted for Publication, 1997: http://poincare.inf.uniroma3.it:8080

[2] S. Abiteboul, S. **Cluet** and T. **Milo.** Correspondence and Translation for Heterogeneous data. In *Proceedings of the International Conference on Database Theory, Greece , 1997.*

[3] E. Bier and A. **Goodisman,** Documents as User Interfaces. EP90, *Proceedings of the International Conference in Electronic Publishing, Document Manipulation & Typography,* R. Furuta Ed., P.249-262, Cambridge University Press, September 1990.

[4] P.J. Brown. A Hypertext System for UNIX. Computing Systems, **2(1):37-53, 1989.**

[5] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Document to Novel Query Facilities. *SIGMOD* 1994.

[6] R. Cattell. The Object Database Standard. *ODMG-93*. Morgan Kaufmann Publishers, ISBN: 1-55860-396-4.

[7] D.N. Duan. Distributed Database Access in a Corporate Environment Using Java *Proceedings of the 5th World Wide Web Conference,* Paris (France), May 1996.

[8] T. Gruber, S. Vemuri, and J. Rice. *Model-Based virtual document Generation.* Knowledge Systems Laboratory, KSL-96-16, May 1996.
http://WWW-KSL-svc.Stanford.EDU:5915/doc/papers/ksl-96-16/

[9] David M. Levy. Document reuse and Document systems. *Electronic Publishing,* Vol.6(4), pages 339-348, December 1993.

[10] Maria Milosavljevic, Adrian Tulloch and Robert Dale. Text Generation in a Dynamic Hypertext Environment. *Proceedings of the 19th Australasian Computer Science Conference,* Melbourne, Australia, January 1996,
http://www.cs.mu.oz.au/conferences96/abstracts/acsc.7c.2.HTML

[11] M. Nanard et al. "Using types to Incorporate Knowledge in Hypertext". *Proceedings of 3rd.ACM Conferences on Hypertext (Hypertext '91),*ACM Press, San Antonio (Texas), pp.329-344, Dec. 1991.

[14] David Skar. Graduating from File-based to Info-based Document Construction. In SGML Asia pacific Conference, Sydney, September 1996.

Web References

[15] DSSSL:
http://occam.sjf.novell.com:8080/dsssl/dsssl96

[16] The Virtual Library. *Conferences:*
http://www.rpd.net/Info/conferences/OVERVIEW-en.HTML

[17] RMIT, *The Structured Information Manager:*
http://www.kbs.RMIT.edu.au/sim/sim.html

[18] Rasmus Lerdorf: *PHP/FI:*
http://www.vex.net/php/

[19] Hughes Technology, *w3-msql:*
http://www.hughes.com.au/product/w3-msql/

[20] Randy Norwood, *What is a document* ?, 24 March 1994
http://www.eden.com/~Ernorwood/document/virtual.htm

[21] 02 Auto-demonstration:
http://www.o2tech.fr:80/o2web/o2webgateway/o2www/auto_l?QuErY%3ATheService-%3Eo2_auto_report(%240)

[22] 02 Server:
http://www.o2tech.fr:80/

[23] Peba:
http://www-comp.mpce.mq.edu.au/msi/peba/

[24] TransactNet, *Wit, The Web Interface Toolkit* :
http://www.transactnet.com/

[25] Inserting objects into HTML, W3C Working draft, 22 April 1996:
http://www.w3.org/pub/WWW/TR/WD-object.HTML

Supporting the Answering Process

M. Wu ***M. Fuller***
Department of Computer Science
RMIT
Melbourne
Australia

ming@mds.rmit.edu.au *msf@mds.rmit.edu.au*

Abstract

This paper is concerned with the way information access systems support the question answering process. This process includes three stages: question formulation, information gathering, and analysis and synthesis. Standard information access technologies are mainly concerned with the second of these stages, providing little or no support for the last stage. However, the raw information gathered at this point can seldom be used directly as an answer. This paper discusses issues relating to the support of the analysis and synthesis stage, and suggests how information access systems might be extended to better support it. This paper also describes a WWW-based experimental interface that permits the evaluation of alternate ways of supporting this analysis and synthesis stage of the answering process.

Keywords *Information Retrieval, Question Answering, Passage Retrieval, Answer Presentation, Hypertext, WWW.*

1 Introduction

Quick, efficient access to stored information is an important part of the modern computer era. Systems designed to store, retrieve, and display collections of documents fall into three main categories: database systems, information retrieval systems, and hypertext systems. Database systems tend to be focused on storage and manipulation of tabular data, rather than on access to text collections. Information retrieval systems, optimised towards archiving and accessing large document collections, tend to be query-oriented ('keyword querying'). Hypertext systems, typically used for smaller, structured collections, support access via browsing ('link querying'). Commonly, information access systems have a blend of these characteristics.

Proceedings of the Second Australian Document Computing Symposium, Melbourne, Australia, April 5, 1997.

The categories mentioned represent alternative approaches to examining stored data. However, the resolution of information needs involves more than just examining collections of documents. The satisfactory resolution of information needs requires that users select evidence and data from the documents that form a collection, that they analyse their content, and that they synthesise their chosen evidence, with the end result being the determination of an 'answer.' Clearly, the portion of this task that is supported by information access systems — the identification and retrieval of sub-portions of larger collections — is only part of the task that confronts a user in search of an answer.

This paper is concerned with the *answering process:* the process that takes place when a user attempts to resolve an information need. In particular, it is about that part of the answering process that occurs after the retrieval of potentially relevant information. The paper explores how properly designed presentation structures can facilitate the analysis and synthesis phases of the answering process. Lastly, a testbed for evaluating such presentation structures is described.

2 Answering Information Needs

2.1 Factors affecting Resolution

There are a number of variables that affect the resolution of users' information needs. These include:

1. the users themselves,
2. the type of their information need,
3. the content of the information collection,
4. the structure and organisation of that content,[1]
5. the ability of the chosen information access system to identify a relevant sub-set of the information collection, and

[1] In the case of hypertext systems, this includes both the distribution of data between data fragments, and the link relationships between data fragments.

6. the method of organising and presenting that content.

The preferences of users and users' knowledge of topic or question can vary. Different users may prefer different approaches to organising and presenting data. For the same information need, different users may have differing levels of foreground or background knowledge with respect to that need. Users may have different levels of experience in dealing with a given collection, and may have varying mental models of its content and organisation.

The motivation and intentions of users also vary with the type of the information need that is to be resolved. Users may have a *specific* need that must be answered, they may need to *learn* about a topic, they may need to *gather* a variety of information, or they may even simply desire to *explore* a collection, without a well defined-goal [3, 25]. Each of these can require different strategies for organising and presenting retrieved data.

Depending on the content and organisation of the data comprising a specific collection or collections, the answer to a user's information need may:

1. not be present at all, or
2. not be present *explicitly*, but be able to be inferred or deduced from information that *is* present, or
3. be present explicitly in a single 'document' or data fragment, or
4. not be present, either explicitly or implicitly, in a single document or data fragment, but be able to be collected from multiple documents or data fragments.

Again, different organisation/presentation approaches may be more or less felicitous in each circumstance.

The structure and organisation of the information collection itself can also have an impact on the resolution of an information need. Are there links present that explicitly indicate relationships between documents? Is there other meta-information available that can be used to guide the resolution of the user's information need? If so, such collection organisation can be utilised by an information access system when it attempts to organise and present the selected data in whatever form is most appropriate for resolving the user's information need.

In the case of those information access systems that select a sub-set of the collection for consideration by a user (whatever the method used), the ability of the system to identify those portions of the collection that are relevant to the user's stated information need is paramount.

The last factor that influences the user's ability to resolve an information need is the choice of method or methods for organising and presenting information from the collection. As can been seen from the preceding paragraphs, this choice can be significantly affected by each of the other factors. Although the organisation and presentation of data could be considered in isolation, it is clear that is better that it be dynamically influenced by an awareness of the differing types of information need, users' idiosyncrasies, the content of a collection (where possible), and the structure and organisation of a collection. It is the assertion of this paper that careful control of the method of organisation and presentation of data can have a marked effect on the ability of an information system to adequately support the resolution of information needs. This issue will be further explored in section 3.2.

2.2 The Answering Process

In the preceding section we saw how both the nature of information needs, and the form of answers to those needs can vary. In this section, we will consider how users go about finding answers, given a particular information need.

Figure 1 shows the different stages of this question answering process. In this process a user begins with an initial question, finds related information, analyses that data, and determines an answer.

The first stage is the identification of the basic information need. The way in which this need is generated, and how information systems support this, is not within the scope of this paper; interested readers are directed to the work of Taylor [21] or Bates [2].

Once an initial question has been formulated, the user can then proceed to gather relevant information, the second stage of the answering process. A browse or query action can be directed at an information system. Such an action is resolved by the system into a set of candidate documents that may potentially be relevant to the user's need. These documents are then prepared in some way by the system for presentation to the user.

The last stage of this process involves the analysis and synthesis of the collected candidate documents. This task involves winnowing the retrieved data to identify the relevant material; the user must analyse each item to determine whether it is potentially an answer component or not. Once this analysis has been performed, the identified answer components must then be gathered and re-organised to synthesise an answer.

Traditional information systems only address the second stage of this process — the collection of potentially relevant material. Database and information retrieval systems typically provide

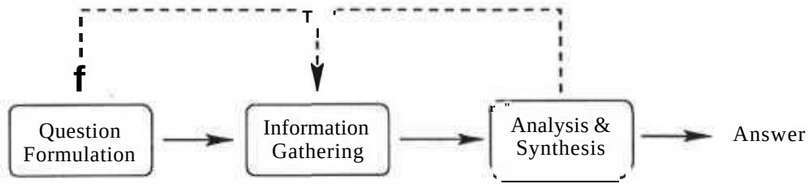


Figure 1: The question answering process

a user with a sorted list of database items that match a specified query. Hypertext systems provide links from one hypertext node to the next, again requiring the user to search for germane material.

To carry out the activities of stage three of the answering process, the user must then sort through the retrieved material, searching for data that can be used to resolve their need. Ideally, however, an information access system should be far more supportive of the requirements of this stage. Rather than simply presenting a sub-set of the collection as being potentially relevant and leaving all further work to the user, information access systems can endeavour to support the analysis/synthesis requirements of the user.

These requirements can be met by changing what it is that such a system presents to users. Analysis of the identified material allows it to be organised in more sophisticated ways. Alternate presentation formats can simplify the cognitive overhead of examining the identified material.

2.3 Information Presentation

Information access systems traditionally present data in several simple ways:

- as a tabular structure (database systems)
- as a linear list of documents (traditional information retrieval systems)
- as a web of inter-linked nodes (hypertext systems)

None of these in their basic form provide any support beyond the information gathering stage of the answering process. For example, a simple linear list of answers is of little use in satisfying browsing needs. A hypertext web is just as inappropriate for fact-finding.

What can be done to aid the selection and analysis task of the third stage of the answering process? Some presentation techniques have been developed that can be useful in this problem. They include:

- Tables of Content — a hierarchical overview of the collection is used to allow users to discern the context of data items, and to navigate between data items.
- Guided Tours — a viewing sequence for relevant documents or nodes is dynamically produced for each query. Unlike most hypertext systems, where an author will have pre-defined all links during or after a hypertext's creation, guided tours are created as the result of a query. Such tours are calculated both with respect to the content of the elements making up the tour, and with respect to the nature of the relationships between those elements [9, 11].
- Indexes — collections are hierarchically organised into categories, allowing users to locate answers by choosing between categories.
- Fish-eye Views — are collection overviews that provide more information about 'near' parts of a collection and less about 'far' parts, thereby supplying both high and low-level information about a collection [10, 7].
- Clusters or Piles — involve using clustering techniques, post-query, to organise and navigate sets of retrieved documents [12, 14], based on the principle that relevant documents tend to be have a greater similarity to each other than to non-relevant documents [22].
- Visualisation Tools — uses the display of information as a matrix of icons, such as query-document similarity [23] or relationships between a variety of document characteristics [16], to help users identify those items most likely to be of interest.

As noted previously, there are a number of variables affecting the resolution of information needs. As these factors change, particularly as the type of the information need changes, so does the appropriateness of each approach to answer presentation [25]. Further, the efficacy of a particular approach may be greater or lesser depending on whether the answer is explicitly present or implicitly present within a specific collection, and on the form and content of that collection.

Of these variations, tables-of-content (displayed with or without using a fish-eye view) and guided tours are based on the internal and external structure of documents, and are best suited to collections of long, well-structured documents. The index, cluster, and visualisation approaches, on the other hand, group documents according to their content or attributes; they are well suited to collections of small to medium size documents, but rather less so to collections of larger, more complex documents such as books or legislative acts.

3 Supporting the Answering Process

The key phase in the process of resolving an information need is that of selecting, analysing, and synthesising data. After a user collects a set of potential candidate documents, she or he may use inter- or intra-document structure to locate interesting material, may compare candidates, may put related parts together, or perhaps arrive at an inferential conclusion. This is the "analysis and synthesis" stage of Figure 1, and it is the result of this phase that can be turned into an "answer."

It is the role of the information system to support these tasks. Its first task is to identify data that is potentially relevant to the user's information need. However, rather than simply baldly presenting it to the user for consideration, it should organise and present the data in such a way that it is easy for the user to evaluate it; the system needs to put the right content in the right form in the right context.

3.1 Appropriate Retrieval Elements

The information that comprises an answer may be found in one or more sentences or paragraphs or other objects in one or more documents. Depending on the content of a collection, answers may be present as either explicit answers or as implicit answers. Explicit answers directly satisfy an information need; implicit answers only inferentially satisfy an information need. In either case, answers to users' information needs may be spread across multiple documents.

The use of the 'document' as the basic unit for information retrieval may be inappropriate for answering questions. Whole documents may be quite unsatisfactory as answer components, on several counts. They may be large and unwieldy; or the components of the answer they contain may be dispersed within the body of the document and be difficult for a user to locate and extract; or the answer components may need to be gathered and digested from multiple documents.

Components of, or evidence for, an answer may occur in just one or a few sentences or paragraphs from a much larger document. Whereas it is conceptually simple to move from viewing such a segment of document to examining the whole, the reverse operation can be extremely difficult: when a document can be several hundred or thousand lines in length, with an arbitrarily complex structure, locating a small part or sentence that contains relevant information is not easy. Even with the provision of such aids as tables of content or indexes, it remains a potentially difficult task. Large documents can also have quite heterogeneous content; some parts may be highly relevant to an information need whilst other parts are highly irrelevant. Smaller *passages* extracted from a document therefore may be preferable as the basic retrieval element [1, 15, 19, 20]; the question of how to identify or select such passages such that the passage is complete, or at least comprehensible, is an open one [4, 5].

3.2 Using Organisation & Presentation to support Analysis & Synthesis

Once a set of potentially relevant passages has been identified, they must be presented to the user. The traditional information retrieval approach is to present them to users one by one. Of the passages that have been identified, some will be completely irrelevant, some will be marginally relevant, and some will be directly relevant. Depending on the collection, there may be varying amounts of redundancy (for instance, consider a collection of newswire article, where a wire story may circulated multiple times in slightly variant, updated forms). Stepping through a long list of such passages places a large information load on users, without providing any hints as to why they were selected, nor where within them answers are to be found.

A simple hypertext browsing approach is no more likely to be better. In this approach, answers are to be discovered by users browsing a hypertext web, traversing links until all of the information that forms an answer has been located. The effectiveness of this very much depends on the user's ability to locate that desired information, and varies with their understanding of their information need, the links that are present within the hypertext, and whatever **ancillary** query mechanisms are available. In the case of a large collection, an important task is that of initially locating the area to be browsed — an entry point. If a collection is too large, or there are too few links, or there are too many links, or the links are the 'wrong' links, then browsing alone may be inadequate to the task; the user will be forced to fall back on some other mechanism, such as querying, to locate the desired information. This brings the problem back to the question of how to organise and present a set of potentially relevant passages.

Two characteristics of passages can be used to help organise them: *content* and *context.*

Related documents are more likely to be more similar to each other than un-related documents [22]. Content can therefore be used to group passages into different *piles,* in the same way people casually organise documents [14, 17]. Passages that contain the same keywords but address different topics are likely to be separated into different groups by this procedure.[2] For example, two users may give the system the same keyword, Panadol, but one user may be a chemist who is interested in Panadol's chemical composition and characteristics, whilst the other is a business analyst who is interested in Panadol's market share. Although the documents in which each is interested may share the same query keywords, clustering techniques can be used to group these passages into different piles. Common index terms can then be used to identify each pile, providing users with some hints as to their content.

Once a pile has been identified by the user as being of interest, in what form and in which sequence should its passages be shown? Can the passages be ranked or organised in some further way?

Passage context can be used to order the presentation of the passages from a given pile [24]. If the source documents are structured, that structural information permits passages to be placed in context to each other, and related passages to be linked. Tables of content can be provided to make such structural relationships explicit. This allows the user to understand the context of each passage with respect to its source document and the collection. In evaluating the rank order, passages that may have a lesser similarity (to the query) but with more links to other passages can be shown in preference to passages with higher similarity but with no or few links to other passages [9, 11]. Such an ordering places an emphasis in locating passages from documents that are highly relevant in part or whole.

At this point, other aids can be provided to support analysis. Guided tours of each pile of candidate documents can be generated according to these guidelines. Tables of content or hypertext maps using fish-eyed views can help provide 'zoom-in' and 'zoom-out' facilities, allowing users to move between a passage-level focus and a document-level focus; the former is likely to be suitable for the learning and exploration types of information needs, the latter for fact-finding and gathering. Visualisation tools can be used to compare passages and indexes to aid further exploration.

The last phase of the answering process is synthesising the answer. A user may have needed to examine multiple passages from multiple documents in order to locate all of the components required to answer her or his question. In order to synthesise that answer, they will need some way of bringing those required components together. Information about passage content and context will still be required by users during this phase to help them organise this material. It is possible that this may be done simply by pulling together the set of user-selected passages, and using the same presentation techniques used during the analysis phase; alternatively, more sophisticated tools may be required to help combine this material into an answer.

4 A Testbed for Answer Presentation

The issues raised in the preceding sections revolve around how variations in presentation format can affect the resolution of information needs, including the question of what granularity of document element is to be preferred as a retrieval element for different information needs.

A testbed interface for exploring these issues has been developed. It uses a suite of CGI scripts to dynamically generate WWW pages, allowing the comparison and evaluation of alternative presentation structures. The interface can track the actions that subjects perform, particularly the choices of presentation formats. For each subject, the following information can be recorded for each question:

- the time at which each subject action occurs.
- the candidate documents that are viewed.
- the presentation formats that are chosen for each document.
- the sequence in which documents are viewed.
- how each document was reached.
- which sentence fragments, paragraphs, or whole documents are marked as "relevant."

Figure 2 shows an example of the interface as viewed using the Netscape browser; this window dump shows a candidate document[3] being viewed

[2] This technique differs from cluster-based retrieval [18], where queries are made against a collection of pre-clustered documents — here, the grouping is made *after* the passages have been identified.

[3] The documents displayed in this figure are taken from the NISS EBSCO MasterFILE database. Information about the MasterFILE database is available at http://www.niss.ac.uk/ebsco/index.html.

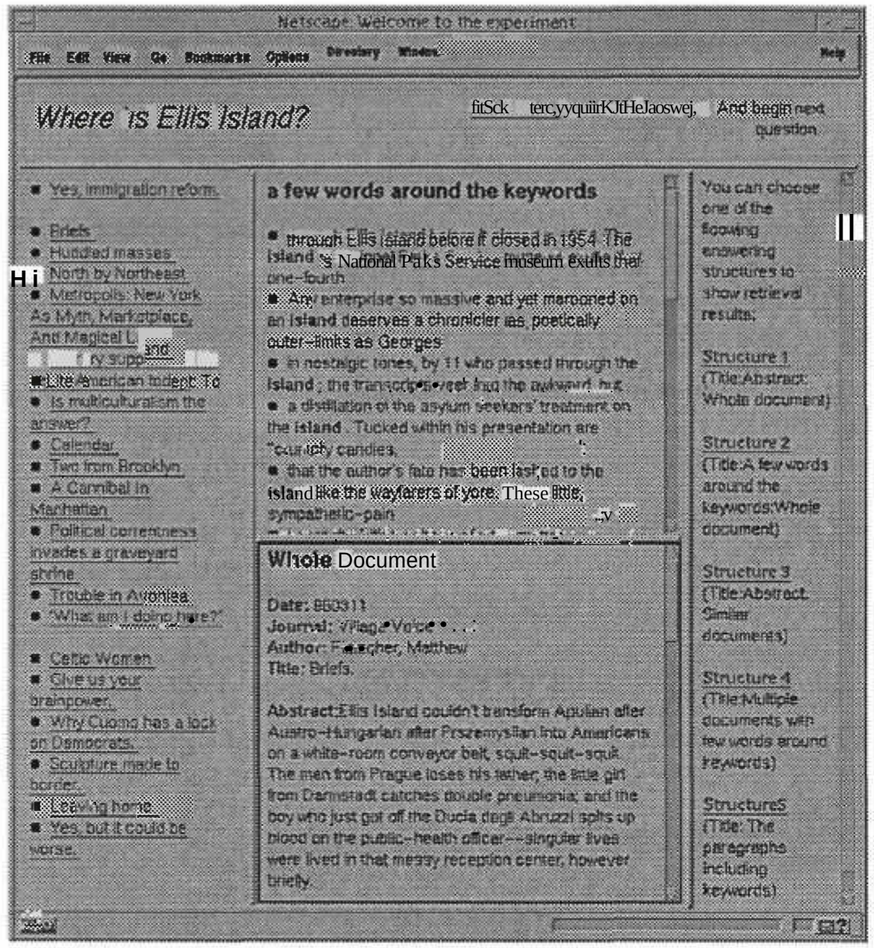


Figure 2: Experiment User Interface

using a presentation format that consists of the set of sentence fragments that contain keywords (in bold) present in the user's query, a separate view of the complete document, plus links to reposition the entire document view to the location of each sentence fragment. In this particular experiment, five alternative presentation formats (displayed on the right-hand side of the example interface) have been selected for analysis. These are:

1. • the document abstract, and
   • the entire document (separate).

2. • the set of **sentence** fragments that contain (highlighted) keywords present in the question, and
   • the entire document (separate), plus
   • links to reposition the document to the location of each sentence fragment.
   (pictured in the middle frames of figure 2)

3. • the document abstract, plus
   • an expandable link to the entire document, and
   • the titles of *n* 'similar' documents.

4. • the sets of sentence fragments that contain (highlighted) keywords present in the question, taken from a user-determined subset of documents, plus
   • expandable links to each of the entire documents.

5. • the set of paragraphs that contain (highlighted) keywords present in the question, plus an expandable link to the entire document.

These alternate presentation formats were chosen to compare the appropriateness of different granularities of document passages for answering different information needs. They also offer a contrast between 'zoom-in' (where the user moves from the whole document to a specific part of it) and 'zoom-out' (from the part to the whole) viewing actions.

Each displayed page in the testbed interface presents to the subject (a) a question (the upper frame), (b) a set of titles of candidate documents (the lower left-hand frame), (c) an area in which a candidate document can be displayed (the lower middle frames), and (d) a list of alternative presentations formats for each candidate document (the lower right-hand frame). By selecting either the title of a new candidate document from area (b), or by selecting a different presentation format from area (d), the subject can control which document is currently displayed in area (c) and which presentation format should be used to display it. When viewing a candidate document or passages from a candidate document, the subject can 'mark' those elements that she or he considers to be relevant to formulating their answer. Once the subject is confident he or she has determined an answer to the current question, she or he can advance to the next question by selecting a link in area (a) (right hand side).

This novel approach has resulted in a experimental testbed that is easy and quick to develop (particularly with respect to standard techniques for developing graphical interfaces). As it is based on a simple collection of CGI scripts, it can be easily extended or modified based on experimental feedback, including adding new or removing old presentation formats. Tracking of user actions is particularly easy. From a usage perspective, the WWW interface is easy to learn, particularly so given that subjects are likely to be familiar with the interface; this reduces subjects' learning curve, allowing them to more rapidly progress into an experimental procedure.

5 Concluding Remarks

This paper has attempted to present our understanding of the question answering process. This process includes three stages: question formulation, information gathering, and analysis and synthesis. Traditional information access systems have been concerned with the second of these stages: the task of locating relevant documents for users. They have little or no support for the equally important analysis and synthesis stage.

The analysis and synthesis stage involves analysing the retrieved material to identify those passages containing answer components, and then synthesising those components into an answer. In order to do so, the user must be able to comprehend the retrieved content. We concur with previous studies that suggest that entire documents may be a poor choice as the basic retrieval element and that smaller passages are likely to be preferable. Exactly what size and granularity for such passages is ideal is however unclear, and may in fact be different for different types of information needs.

The simple list-based presentation of traditional information retrieval systems has similarly been show to be inadequate for analysing those passages that are identified as potentially relevant. We suggest that a two-level approach to presentation that draws on several different techniques will prove efficacious. This approach would firstly organise retrieved passages using the 'piles' metaphor, and then use structural information to further organise passages within each pile. When presenting the content of a pile, other techniques such as tables of content, indexes, and visualisation tools can be used to enhance the user's identification and understanding of each passage. A similar approach may be appropriate during the synthesis phase of the question answering process.

Many factors influence the design of presentation structures. Important variables include the users themselves, the type of the user's information need, the content and structure of each document, and the content and structure of the collection. It is not clear which structures are most suitable in which circumstance, or if there is a generic structure that can be used.

To help resolve these questions, a novel testbed has been developed. It uses CGI scripts to automatically and dynamically generate WWW pages that can be used in experiments to compare and contrast the utility of alternate presentation structures in the resolution of different information needs. This testbed is interesting in that it is highly suitable for the rapid development and modification of experiments, and results in an interface that is easy to use and is also likely to be familiar to experimental subjects. We will be using this testbed to explore these and other questions related to information discovery and the answering process.

Acknowledgements

Part of this work was supported by an A.R.C. grant.

The authors would like to thank James Thorn and Ross Wilkinson for their advice and input.

References

[1] Suliman Al-Hawamdeh and Peter Willett. Paragraph-based nearest neighbour searching in full-text documents. *Electronic Publishing*

— *Origination, Dissemination, and Design*, Volume 2, Number 4, pages 179-192, December 1989.

[2] Marcia J. Bates. The design of browsing and berrypicking techniques for the online search interface. *Online Review*, Volume 13, Number 5, pages 407-424, 1989.

[3] N. J. Belkin, P. G.. Marchetti and C. Cool. Braque: Design of an interface to support user interaction in information retrieval. *Information Processing and Management*, Volume 29, Number 3, pages 325-344, 1993.

[4] James P. Callan. Passage-level evidence in document retrieval. In Croft and van Rijsbergen [6], pages 302-310.

[5] Marie Louise Corral and Josiane Mothe. How to retrieve and display long structured documents? In *Proceedings of the Basque International Workshop on Information Technology*, pages 10-19, San Sabastian, Spain, July 1995. IEEE Computer Society.

[6] W. Bruce Croft and C. J. van Rijsbergen (editors). *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin University.*, Dublin, Ireland, 3-6 July 1994. Springer-Verlag.

[7] D. E. Egan, J. R. Remde, L. M. Gomez, T. K. Landauer, J. Eberhardt and C. C. Lochbaum. Formative design-evaluation of SuperBook. *ACM Transactions on Office Information Systems*, Volume 7, Number 1, pages 30-57, January 1989.

[8] Hans-Peter Frei, Donna Harman, Peter Schäuble and Ross Wilkinson (editors). *Proceedings of the 19th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval.*, Zurich, Switzerland, 18-22 August 1996. ACM.

[9] M. E. Frisse. Searching for information in a hypertext medical handbook. In *Proceedings of the ACM Hypertext '87 Conference*, pages 57-66, University of North Carolina at Chapel Hill, November 13-15 1987. **ACM.**

[10] G. W. Furnas. Generalized fisheye views. In *CHI'86 Proceedings*, pages 16-23, April 1986.

[11] Catherine Guinan and Alan F. Smeaton. Information retrieval from hypertext using dynamically planned guided tours. In J. Nanard, M. Nanard and P. Paolini (editors), *Proceedings of ECHT'92 the Fourth ACM Conference on Hypertext*, pages 122-130, Milano, Italy, November 30 - December 4 1992. ACM.

[12] Marti A. Hearst and Jan O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In Frei et al. [8], pages **76-84.**

[13] R. Korfhage, E. Rasmussen and P. Willett (editors). *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Pittsburg, U.S.A., June 27 - July 1 1993. ACM.

[14] Richard Mander, Gitta Saloman and Yin Yin Wong. A 'pile' metaphor for supporting casual organisation of information. In *Human Factors in Computer Systems: CHI'92 Conference Proceedings*, pages 627-634, Monterey, California, May 3-7 1992. ACM Press.

[15] A. Moffat, R. Sacks-Davis, R. Wilkinson and J. Zobel. Retrieval of partial documents. In *The Second Text REtrieval Conference (TREC-2)*, Gaithersburg, Maryland, March 1994. National Institute of Standards and Technology. Special Publication 500-215.

[16] Lucy Terry Nowell, Robert K. France, Deborah Hix, Lenwood S. Heath and Edward A. Fox. Visualizing search results: some alternatives to query-document similarity. In Frei et al. [8], pages 67-75.

[17] Daniel E. Rose, Richard Mander, Tim Oren, Dulce B. Poncelon, Gitta Saloman and Yin Yin Wong. Content awareness in a file system interface: Implementing the "pile" metaphor for organising information. In Korfhage et al. [13], pages 260-269.

[18] G. Salton. *Automatic Text Processing.* Addison-Wesley, Reading, Massachusetts, 1989.

[19] G. Salton, J. Allen and C. Buckley. Approaches to passage retrieval in full text information systems. In Korfhage et al. [13], pages 49-58.

[20] Gerard Salton and Chris Buckley. Automatic text structuring and retrieval - experiments in automatic encyclopedia searching. In *Proceedings of the 14th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 21-30, Chicago, U.S.A., 1991. ACM.

[21] Robert S. Taylor. The process of asking questinos. *American Documentation*, pages 391-396, October 1962.

[22] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, London, 1979.

[23] Aravindan Veerasamy and Nicholas J. Belkin. Evaluation of a tool for visulisation of information retrieval results. In Frei et al. [8], pages 85-92.

[24] R. Wilkinson. Effective retrieval of structured documents. In Croft and van Rijsbergen [6], pages 311–317.

[25] Ross Wilkinson and Michael Fuller. Integrated information access via structure. In Maristella Agosti and Alan Smeaton (editors), *Information Retrieval and Hypertext*, pages 257-271. Kluwer Academic Publishers, 1996.

Collection Selection via Lexicon Inspection

Justin Zobel
Department of Computer Science, RMIT
GPO Box 2476V, Melbourne 3001, Australia
jz@cs.rmit.edu.au

Abstract

A distributed text database consists of multiple individual text collections. When a query is posed to a distributed text database, significant computational resources can be saved by identifing the individual collections that are the most likely to contain answers, as unnecessary accesses to the other collections will be avoided. In this paper we explore the potential of one approach to selecting collections: ranking them according to the content of each collection's lexicon. We outline principles on which such ranking might be based and how its performance can be evaluated. Experiments with two sets of text collections show that use of lexicons to select collections can be effective, but depends on how performance is measured.

1 Introduction

Text databases can be distributed across a set of individual text collections. For example, a single site might have several single-theme text databases, each, say, a set of newspaper articles or a collection of documents on a particular topic; a user may well issue a query to the set of such collections as well as to the collections separately. Another example of a distributed text database is a suite of collections located at different remote sites that are regarded as a single database; an instance is a geographically distributed company that maintains records locally at each site. There are clear advantages to providing a front-end query processor—a *receptionist*—that provides transparent access to such sets of collections. Not only does a receptionist save the user the effort of querying each collection independently, but there are potential efficiency gains if the receptionist can identify collections that are unlikely to contain answers.

The problem of identifying likely collections is particularly difficult in the context of ranking queries. Such queries—in which the information need is expressed informally in natural language or by a list of keywords—are known to be at least as effective for document retrieval as traditional Boolean queries [9]. (In the context of a text database, effectiveness is measured by the ability of a query language and query evaluation technique to identify documents that a user judges to be relevant to the information need.) But for a ranked query there is no precise set of documents that are known to be answers, since the full set of documents in the collection is evaluated for statistical similarity to the query and then ranked by this similarity: all the documents are, to some degree, matches.

There are several approaches that might be used to determine the relevance of a collection to a query. Given unlimited resources the receptionist could fully index every document in every collection [8], but such an approach is not always practical. The question then becomes: what information can the receptionist afford to store about each collection, or what information can the receptionist cheaply extract from each collection in response to a query? An obvious answer is the lexicon of each collection, that is, the set of distinct indexed words. For collections of at least moderate size the lexicon is typically small, at perhaps 1% or less of the total data size; by way of contrast an elementary index is typically around 10% of the total data size [7]. For example, the lexicon of the first 3 Gb of the TREC data [4] is roughly one million words and occupies under 20 Mb, including space for two frequency counts for each word. If even this small requirement is unacceptable, given a query it is relatively cheap (compared to the total cost of query evaluation) to poll each subcollection to extract statistics about each query term.

However, although this information is cheap to extract or collect, there are obvious limitations to its usefulness. It can tell us how many documents in a collection contain a given query term, but not, for example, whether any document in a collection contains any particular combination of query terms, or even whether they co-occur in any document at all.

In this paper we explore the potential of lexicon inspection to successfully select useful collections. If effective, lexicon inspection would give considerable efficiency gains by allowing a query evaluator to simply ignore part of the database. First we analyse the task of collection selection, to identify the basis on which a collection should be ranked highly. Given this basis we consider how lexicons might be used to rank collections, and how the performance of a ranking method might be evaluated. We then describe the results of our experiments with collection selection, which compare a variety of ranking formulations on collections extracted from the TREC data. These experiments show that a surprisingly simple formulation—the "inner product"—can be effective; however, not only is there considerable scope for improvement, but different measures of effectiveness produce quite different results. Whether improvement is achieveable within the limitations of lexicon inspection is an open question.

Collection selection is one problem in the larger question of query evaluation in distributed text database systems; other important problems include management of communication costs and comparison of similarities from separate collections, that is, the collection fusion problem [1, 8, 11]. These problems are not considered in this paper.

2 Collection selection

In a distributed text database system the documents retrieved in response to a query should be the same as would be retrieved were the documents stored monolithically, because monolithic retrieval systems are known to be reasonably effective. That is, the aim of the receptionist is to emulate a monolithic system, so that from the user's perspective the distribution is transparent. However, efficiency considerations may prohibit an exact emulation: centralised indexing requires significant additional space (and has practical drawbacks with regard to update and consistency); while evaluating a query against every subcollection in a distributed system is expensive.

In a monolithic system, documents are ranked against a query using a *similarity measure* that computes the statistical similarity of each document to the query. The set of documents can then be ordered by similarity, giving a ranking. One of the most effective similarity measures is the cosine measure [2, 9, 10], which can be formulated as

$$C(q,d) = \frac{\sum_{t \in q \& d} w_{q,t} \cdot w_{d,t}}{W_q \cdot W_d}$$

where q is the query, d is the document,

$$W_x = \sqrt{\sum_{t \in x} w_{x,t}^2},$$
$$w_{q,t} = w_t \cdot \log(f_{q,t} + 1),$$
$$w_{d,t} = \log(f_{d,t} + 1),$$
$$w_t = \log(N/f_t + 1),$$

the value W_x is the length of query or document x, the value $w_{x,t}$ is the importance of t in x, the value $f_{x,t}$ is the frequency of term t *in* x, the value w_t is the overall importance of term t, the value N is the number of documents in the collection, and f_t is the number of documents containing t.

However, even the best similarity measure will highly rank documents that are not relevant to a query; high similarity does not guarantee that a human will judge the document to be pertinent. We are thus presented with two definitions of "answer", based on similarity on the one hand and relevance on the other.

In evaluating the performance of lexicon selection it is necessary to choose between these definitions of answer: should we select collections that contain highly similar documents or collections that contain relevant documents? If the aim is to emulate a retrieval mechanism the former choice is appropriate, as argued by Gravano and Garcia-Molina [3]. However it is not clear that this is the best strategy. Consider a distributed database consisting of a large number of small subcollections; an example is a database in which each subcollection is a website. For a given query a particular subcollection might be primarily about the topic of the query, and thus highly relevant; but this does not guarantee that all the documents in the subcollection will be highly ranked. Moreover, the receptionist may be no more than a gateway to a set of subcollections, in which case the exact retrieval method used in a subcollection may be unknown. (This aspect of the collection fusion problem is considered by Voorhees et al. [11].) Thus choice of what to emulate is, in effect, a design parameter for a distributed text retrieval system.

Moreover, it is plausible to suppose that in the context of the TREC data used in the experiments below the two definitions of answer may be indistinguishable. Each TREC disk contains some hundreds of thousands of documents, of which, for each query, only a few thousand are manually judged for relevance. These documents are judged because they have been highly ranked according to some similarity measure. Thus a document can only be known to be relevant if it is highly ranked, and—because similarity measures are considerably better than random at identifying relevant documents—highly ranked documents have a reasonable likeli-

Proceedings of the Second Australian Document Computing Symposium, Melbourne, Australia, April 5, 1997.

hood of relevance. In this paper, however, we have used both mechanisms to measure the effectiveness of collection selection.

3 Lexicon inspection

A similarity measure such as cosine is effective at ranking documents because it embodies the following principles. A document will be highly ranked by the cosine measure if: it contains many of the more important query terms; the query terms are frequent in the document; and the document is dense with the query terms, hence the importance of normalising by document length.

A subcollection can be regarded as a single document, and indeed for the purposes of ranking is described by its lexicon, which contains the distinct words of the subcollection and their frequencies of occurrence. The problem is then identification of an appropriate ranking formulation.

A natural first choice is to investigate similarity measures such as cosine, since they are effective in other contexts. But two of the three principles of effective document ranking are not particularly applicable to lexicon ranking: it is probably not desirable to normalise by subcollection size, if only because, statistically, a larger subcollection is more likely to contain relevant documents; and, for a set of subcollections of reasonable size, most of the lexicons will contain most of the query terms. (For highly specific query terms such as proper names this will not be true, but an effective collection selection mechanism must rank lexicons even when they all contain all of the query terms.) Moreover, a little more information is available in the context of collection selection: we know both the frequency of each term in each collection and the number of documents with each term in each collection. That is, the most important available statistics are the number C of subcollections, the number N of documents, the number N_c of documents in each subcollection, and, for each query term t: the number f_t of documents containing t, the number F_t of occurrences of t, the number $f_{c,t}$ of documents containing t in collection c, and the number $F_{c,t}$ of occurrences of t in c.

Thus a subcollection should be highly ranked if, for each query term,

- The query term occurs in the subcollection.
- The query term is common in the subcollection, or rather, common relative to the other subcollections.
- The subcollection contains a relatively high proportion of documents with the query term.
- There are likely to be documents in the subcollection in which the term is relatively frequent.

The cosine measure could in this context be defined by

$$C(q,c) = \frac{\sum_{t \in q\&c} w_{q,t} \cdot w_{c,t}}{W_c} \text{ where} \quad (1)$$

$$w_{c,t} = \log(f_{c,t} + 1)$$

and the other symbols are as defined earlier. But as discussed above this formulation would not be expected to be particularly effective; and indeed it was found to be unsuccessful in preliminary work on this problem by Moffat and Zobel [6]. Another possible similarity measure is the inner product—a standard ranking formulation used when document length is not available, and potentially more effective for collection ranking.

$$I(q,c) = \sum_{t \in q\&c} w_{q,t} \cdot w_{c,t} \text{ where} \quad (2)$$

$$w_{c,t} = \log(f_{c,t} + 1) \cdot w_t$$

Other work has identified the inner product as an effective measure for collection selection [3].

However, it should also be possible to make productive use of more of the information listed above. One obvious test is to replace f_t by F_t and $f_{c,t}$ by $F_{c,t}$, but in preliminary experiments this was not successful. A more interesting experiment is to attempt to estimate whether the query terms are atypically frequent in the collection. This property, the *skew*, showed a limited improvement in experiments with document retrieval [12]. One formulation of a similarity measure based on skew is

$$S(q,c) = \sum_{t \in q\&c} \frac{f_{c,t}}{f_t} \cdot f_{q,t} \cdot w_t \quad (3)$$

Another approach is to use the cosine measure to estimate the highest-available similarity for a typical-length document in the subcollection. A possible formulation is

$$H(q,c) = \frac{\sum_{t \in q\&c} w_{q,t} \cdot w_{c,t}}{W_c} \text{ with} \quad (4)$$

$$w_{c,t} = \log(F_{c,t} + 1) \cdot w_t \text{ and}$$

$$W_c = \sqrt{\frac{\sum_{t \in c} F_{c,t}}{N_c}}$$

In W_c the expression $\sum_{t \in c} F_{c,t}/N_c$ is the average number of term occurrences in each document in the collection, and the square root of the number of term occurrences provides a reasonable approximation (for ranking purposes) to more formal statements of document length [13].

4 Experimental results

Test data

We used two sets of test data to evaluate collection-selection techniques. The first was disk 2 of the TREC data [4]. As part of the TREC "collection fusion" track the data on this disk has been partitioned into 43 subcollections, which vary from around 1,600 to 7,500 documents each. (However, the number of word occurrences in each collection varies rather less, at close to three million words each. We do not believe that this invalidates the results. A problem that may introduce bias is that the longest documents, from the Federal Register, are rarely relevant, thus favouring measures that highly rank collections with more documents.) Each subcollection is drawn from a single source, such as one month of articles from the Wall Street Journal. The query set used is queries 51 to 150, which have around 15,000 known relevant documents in this data; these queries have an average of just over one hundred word occurrences each.

However, this data set was used for developing the measures, and thus does not provide an independent evaluation of their performance. We thus chose a second data set with different characteristics. We again used TREC data, this time disk 3, but divided the data into 91 subcollections by randomly choosing points at which new subcollections began; the number of documents per subcollection varied from 14 to almost 23,000, with an average of around 5,000. The query set used was queries 202 to 250, which have an average of just under ten word occurrences each and are thus much shorter than the queries used with the first data set.

For both sets, data and queries were preprocessed with the Lovins stemmer [5].

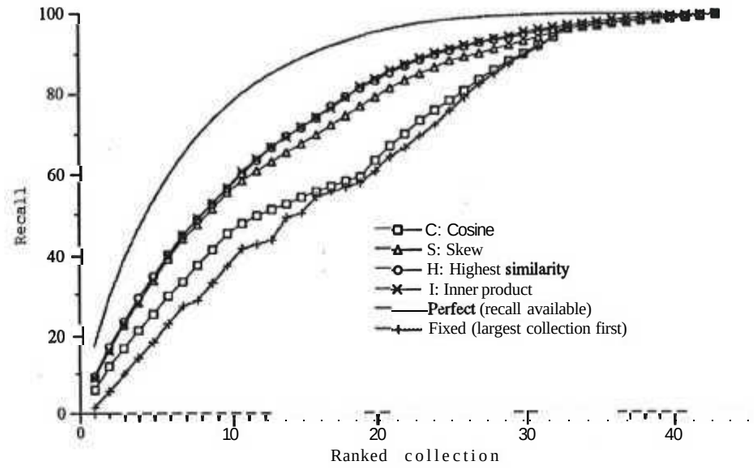


Figure 1: *Performance of similarity measures for collection selection on TREC disk 2, measured by ability to identify collections with known relevant documents.*

Experiments

For each query, the similarity measures described above give a ranking of subcollections. There are several possible methods for evaluating these rankings. One is to count the number of relevant documents occurring in the highest k ranked subcollections, for, with the first data set, k from 1 to 43. This measures the effect of using a receptionist that ranks the subcollections and returns documents from the first k; it measures how many of the relevant documents the user will have an opportunity to see, that is, the potential recall. Another possible evaluation method is examine the proportion of documents in the k highest-ranked collections that are relevant. This measures how dense the first k collections are with relevant documents.

These evaluation methods have the potential to produce rather different results (consider for example the effect of having a small collection in which every document is relevant), but for the test data used here the methods were equivalent: they produced the same ranking of similarity measures. The reported results are based on the first method, in which the ideal ranking has the subcollections ordered by decreasing number of relevant documents.

To bound the performance of the similarity measures we also considered two baselines. One was the best-available performance: the *perfect* ranking given by, for each query, ordering the collections by decreasing number of relevant documents. The other baseline was the query-

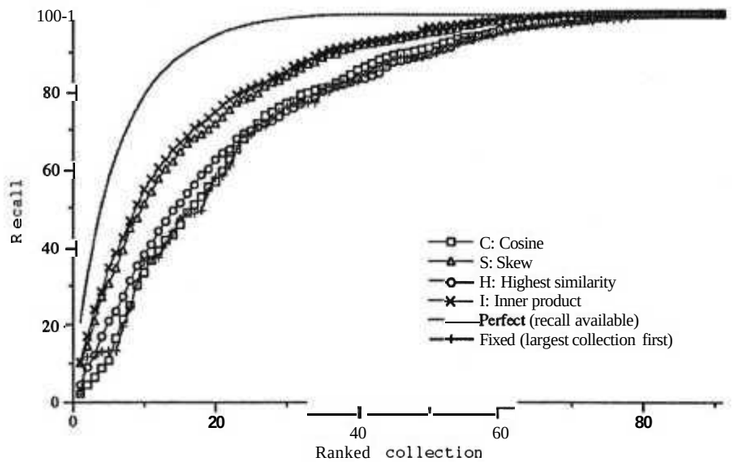


Figure 2: *Performance of similarity measures for collection selection on TREC disk 3, measured by ability to identify collections with known relevant documents.*

independent, *fixed* ranking given by ordering the collections by decreasing size.

Results for the first data set are shown in Figure 1. The graph shows how many subcollections must be inspected to achieve a certain level of recall. The best line is the perfect ranking and the worst is the fixed ranking. All of the similarity measures have done at least as well as the fixed ranking, showing that for this data lexicons can be used to select collections. The fixed ordering is itself above the diagonal of the graph, showing that it is rather better than a random ordering of the subcollections.

The cosine measure has, as predicted, not worked well, but the other three measures—the most successful of a large number of variants tested—have done rather better. Indeed, their performance is remarkably similar; function *S* is marginally the weakest of the three, function *H* achieves slightly higher precision (with more relevant documents in the first six subcollections), and function *I* is best overall. All three manage to order the subcollections such that the first 10 contain just over 50% of the relevant documents.

Gravano and Garcia-Molina [3] have argued that inner product works well for collection selection. What the results on disk 2 show is that inner product does considerably better than fixed ordering, which was not clear from the results of Gravano and Garcia-Molina; and that, with the information available, it is difficult to do better. In principle, better performance is available; but it is most unlikely that anything like the best performance can be achieved in practice, just as it is impossible to design a similarity measure for ranking documents that achieves 100% recall precision.

However, this data set was used to develop and test the similarity measures used for collection selection (other than inner product, which was already thought to work well). We used independent experiments to confirm that these similarity measures are indeed effective. The results of these experiments, on the randomly-partitioned disk 3, are shown in Figure 2. The preliminary results are broadly confirmed; the higest-rank measure has done badly but both inner product and skew have been effective. The problem of collection selection is undoubtably more difficult in this case, both because the subcollections vary more in size and because the shorter queries provide less evidence to the ranking mechanism. Nonetheless lexicon ranking appears considerably better than the fixed ordering.

We also used the alternative method for measuring lexicon ranking. The TREC relevance judgements are formed by a pooled method, in which the top 100 documents returned by each of a range of mechanisms is manually assessed. Thus the complete list of assessments—relevant and non-relevant—is of documents that some similarity measure has ranked highly. We can therefore use this list to measure the ability of our lexicon similarity measures to identify collections that contain highly ranked documents. The results of this experiment on disk 3 are illustrated in

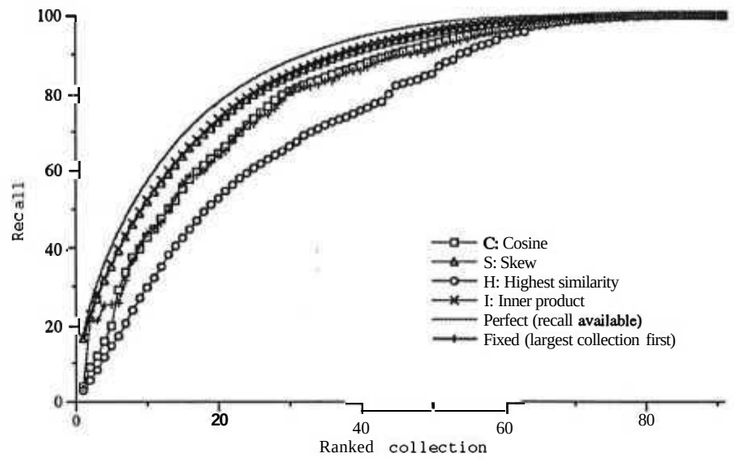


Figure 3: *Performance of similarity measures for collection selection on TREC disk 3, measured by ability to identify collections with highly ranked documents.*

Figure 3. Of the similarity measures both inner product and skew have done well, coming close to the perfect performance and placing the first 50% of highly ranked documents in the first 10 of the 91 subcollections; but note that even the fixed ordering places 45% in the first 10 subcollections.

However, applying this methodology to the first data set, disk 2, yields the distinct surprise illustrated in Figure 4. Despite the positive results for this data set illustrated in Figure 1, none of the similarity measures has done any better than the fixed ranking—indeed, they have done slightly worse than a random ordering. These results contradict our intuition that success according to one measure will be reflected by success according to the other.

5 Summary

In any distributed text collection—be it search engines for the Web or a more formal federation of text databases—it would be valuable to have some mechanism for identifying the subcollections that are most likely to contain answers. Ideally this mechanism should avoid both the storage overheads of full indexing of the subcollections and the processing costs of querying every subcollection at every query.

In this paper we have explored the use of lexicons for selecting collections. We identified appropriate baselines for comparison, perfect ordering and a fixed ordering, and possible approaches to measurement of performance. In both our experiments and in other published work it appears that simple similarity measures can be applied to ranking of lexicons. However, the degree of success appears to depend on what is measured: in our experiments lexicon ranking was better at identifying collections with relevant documents than at identifying collections with highly-ranked documents. It is the use of appropriate baselines that makes evident the failure according to the second measure; although earlier work using this measure [3] was optimistic it is not clear that in those experiments lexicon ranking was significantly more effective than fixed orderings such as largest collection first.

However, it is clear that lexicon ranking leaves considerable scope for improvement. The imperfection of lexicon selection as a direction for collection selection means that further research is required—identification of effective collection selection techniques will require exploration of more radical alternatives.

Acknowledgements

I am grateful to Daryl D'Souza, Alistair Moffat, and James Thorn. This work was supported by the Australian Research Council.

References

[1] D.J. D'Souza and J.A. Thom. How good are similarity measures across distributed document collections? In *Proc. Australian Document Computing Conference*, pages 63-66, Melbourne, Australia, March 1996.

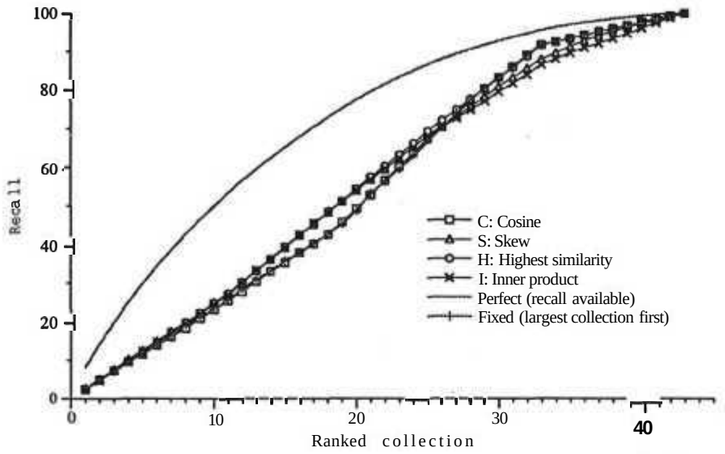


Figure 4: *Performance of similarity measures for collection selection on TREC disk 2, measured by ability to identify collections with highly ranked documents.*

[2] W.B. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, 1992.

[3] L. Gravano and J.H. Garcia-Molina. Generalising GlOSS to vector-space databases and broker hierarchies. In *Proc. International Conference on Very Large Databases,* Zurich, Switzerland, 1995.

[4] D.K. Harman. Overview of the first Text Retrieval Conference. In D.K. Harman, editor, *Proc. TREC Text Retrieval Conference,* pages 1–20, Washington, November 1992. National Institute of Standards Special Publication 500-207.

[5] J.B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computation,* ll(l-2):22-31, 1968.

[6] A. Moffat and J. Zobel. Information retrieval for distributed text database systems. Manuscript in submission.

[7] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems.* (To appear).

[8] A. Moffat and J. Zobel. Information retrieval systems for large document collections. In D. Harman, editor, *Proc. Text Retrieval Conference (TREC),* pages 85-93, Washington, 1994. National Institute of Standards and Technology Special Publication 500-225.

[9] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, Reading, MA, 1989.

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

[11] E.M. Voorhees, N.K. Gupta, and B. Johnson-Laird. Learning collection fusion strategies. In *Proc. ACM-SIGIR International Conference on Research and Development in Information Retrieval,* pages 172-179, Seattle, Washington, July 1995.

[12] P. Wallis, J. Zobel, and J.A. Thorn. Document ranking, topic, and skew. In *Proc. Japan-Australia Joint Symp. on Natural Language Processing,* pages 325-332, Iizuka City, Japan, 1991.

[13] J. Zobel and A. Moffat. Similarity measures explored. Technical Report TR-95-3, Collaborative Information Technology Research Institute, RMIT and The University of Melbourne, 1995.