Extending SGML to accommodate database functions:
A Methodological Overview

Arijit Sengupta[*]
Computer Science
Indiana University
asengupt@indiana.edu

Andrew Dillon
School of Library & Information Science
Indiana University
adillon@indiana.edu

Abstract

A method for augmenting an SGML document repository with database functionality is presented. SGML [ISO 8879, 1986] has been widely accepted as a standard language for writing text with added structural information that gives the text greater applicability. Recently there has been a trend to use this structural information as meta-data in databases. The complex structure of documents, however, makes it difficult to directly map the structural information in documents to database structures. In particular, the flat nature of relational databases makes it extremely difficult to model documents that are inherently hierarchical in nature. Consequently, documents are modeled in object-oriented databases [Abiteboul et al., 1993], and object-relational databases [Holst, 1995], in which SGML documents are mapped into the corresponding database models and are later reconstructed as necessary. However, this mapping strategy is not natural and can potentially cause loss of information in the original SGML documents. Moreover, interfaces for building queries for current document databases are mostly built on form-based query techniques and do not use the "look and feel" of the documents. This paper introduces an implementation method for a complex-object modeling technique specifically for SGML documents and describes interface techniques tailored for text databases. Some of the concepts for a Structured Document Database Management System (SDDBMS) specifically designed for SGML documents are described. A small survey of some current products is also presented to demonstrate the need for such a system.

1. Introduction

Standard Generalized Markup Language (SGML) [ISO 8879, 1986] has been widely accepted for writing documents for "universal" representation. SGML was originally designed as a way to build platform and system-independent documents. SGML by itself does not give any semantics to documents. It is a means for embedding logical structure information into documents that can be later used by applications to insert formatting information based on the embedded structure. Since the logical structure is completely independent of the platform or software on which it is created, there is no problem transferring the original document to any platform. The applications in the target platforms can then act as appropriate depending on the system and platform and apply system-dependent formatting to the document for printing, displaying or exporting to other forms.

The information added to text using SGML is powerful enough to deliver many other useful functions. The most prominent among these is the ability to search a document based on content and structure. Current word-processor documents only offer simple string search facilities. The addition of structural information in the document allows the user to incorporate this structure in his/her searches, thus giving him/her the ability to perform very powerful searches, formally called queries.

2. The Need for Querying

There are many reasons why the concept of SGML databases is a good idea. Primary among these is the ability to query the documents. Although there are many systems in the market for authoring and viewing documents in SGML, few of these products support complex queries, and most only have the capability to search for simple text strings. Some products have the option of limiting the searches by SGML regions or tags. However, very few can combine these searches with other query constructs such as *join*, *negation*, *quantification* to provide a sophisticated query facility. These querying capabilities could prevent users from *getting lost in hyperspace*, a common problem with complex databases [Hammond and Allinson, 1989].

[*] Partially supported by US Dept. of Education award number P200A502367 and NSF Research and Infrastructure grant, award number NSF CDA-9303189.

HyperText Markup Language (HTML), the popular language for the World Wide Web (WWW), has recently been adapted as an application of SGML [HTML Working Group, 1995]. We utilized this fact to perform a small experiment to understand the need for querying capabilities in SGML hypertext systems. To accomplish this, we put together two sets of SGML documents of differing sizes and subject matters with a hypertext navigation and search facility. All of the documents were marked up in SGML and indexed with Open Text's software [Open Text, 1994]. We built web-based interfaces for browsing as well as searching the documents. Although browsing is the most popular (and often, the only) means for traversing through documents in the web, interest in automated searches was manifest in the users studied here.

One of the document sets used in this experiment was the English Poetry Full-Text Database from Chadwyck-Healey [Chadwyck-Healey, 1994]. Since these documents were not organized in a form suitable for comfortable browsing, we generated some index-like navigational structures from the embedded information in the documents. We also converted individual poems from the internal SGML format to HTML to make them viewable in any WWW browser. This approach let users navigate the documents by periods, classified by authors and their works. Every navigation level had navigational cues for jumping to the searching mode and back. We found that on average, about 20 percent of the users jumped to the search pages from various levels of navigation [Figure 1].

Figure 1 depicts the navigation pattern for one of the experiments. In this experiment, we used the Chadwyck-Healey English Poetry Database with a navigation path through various periods, poets in those periods, and poems written by the poets. The search pages provided a form-based interface for expressing search clauses using pre-chosen fields, and combinations of such search clauses.

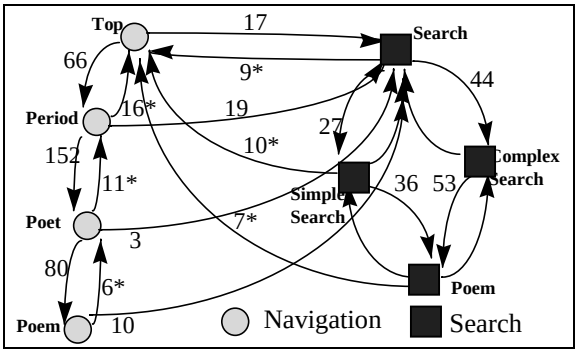

Figure 1. Navigation patterns from browsing to searching modes

In the above figure, the subtree on the left with circular nodes represents pages with explicit navigational cues for going to the next poet, next poem of the poet, and so on. The node marked "Top" represents an index page with the various periods of English Poetry that the database coves. The node marked "Period" lists poets in specific periods. The node marked "Poet" represents pages that contain lists of poems written by specific poets, and the node "Poem" represents the actual poems. Each node actually represents all the pages in the same level. So the "Poem" node represents all the poem pages.

The search subtree on the right is shown with square nodes to differentiate it from the navigation subtree. This subtree contains one main page representing the search interface which has two components: (i) a simple search component in which the user simply enters a single word or phrase to search for, and (ii) a complex search component in which the user can specify multiple search conditions limited by different fields and combined with logical operators. The two intermediate nodes marked by "Simple Search" and "Complex Search" represent the lists of poems returned by the respective searches. The bottommost node "Poem" represents the actual poem, as before.

The labels on the arrows in Figure 1 shows the number of accesses obtained from a small experiment performed with the poetry pages. We collected the data from the access-log generated by the WWW server over a period of one week after making a public announcement of the availability of these pages in a university newsgroup. In every phase of the navigational route, users had options to jump to the search route or continue navigation. The arrows indicate the direction of navigation and are labeled by the number of accesses in that navigation path. For example, 17 accesses were logged from the top level page directly to the search page -- indicating the number of people who preferred to directly access the search interface without bothering to browse around. The apparent anomaly of more accesses in lower levels can be explained by the back-and-forth navigation. For example, users go back to the period level from the poet level, and then access a different poet from there. Numbers with '*' indicate back transitions that may not be accurate because of the caching implemented by browsers. This explains why only 11 back transitions were recorded from the poet level to the Period level, since for other transitions, browsers simply showed the period page that was already cached. Back transitions to auto-generated pages, such as the simple and complex search indices, are not labeled.

Because of the copyrighted nature of the database used, we can not make these pages publicly available. However, the Top two levels of the navigation pages and the search pages are made available for public access [See Poetry1, 1995 for the actual WWW addresses]. The results from the search functions are, however, not publicly available.

These data show the importance of querying in structured documents. The normal string searches provided by the browsers were not included as part of the search path. We argue that automated queries are useful to users of hypertext database systems. The fact that about 20% of the people who used the network of hypertext documents switched over to the search mode shows that some hypertext users would be willing to revert to searching if such a facility existed. The experiment also shows user-preferences and the types of queries that are used most often. The data are valuable for building the optimization algorithms targeted towards these queries.

2.1 "Difficult" Queries

The easiest queries to perform in most hypertext systems are simple string searches. Anything more complicated requires special software, such as the applications described in Section 3. This paper concentrates on queries that require text as well as structural information. Full-text queries or regular expression queries are not given major importance. Some of the types of queries that we considered important in such scenarios are the following:

- Select all reports that have **more than ten** sections (*aggregation*)
- Select all the articles in a journal so that **there exists** an article written by "Dr. Charles Goldfarb" which is referenced in it (*quantification*)
- Select books that have a chapter **title** which **is same as the title** of the first chapter in "Practical SGML" (*join*)
- Select the articles which have **no footnotes** (*negation*)

These types of queries are routinely performed in relational databases. The ability of relational databases to perform these types of queries makes them popular and versatile for various types of applications. The ability to perform these queries in structured documents will likely increase the applicability of documents manifold. However, implementing these queries in the context of text databases requires a lot of effort.

3. The State of The Art

SGML systems have been evolving since 1986, when the standard was first published [ISO 8879, 1986]. Currently, most of the popular word-processing and publishing systems have the capability of publishing in SGML. Word processors can incorporate an SGML Document Type Definition (DTD) into their "styles" structure, thus allowing users to author directly in SGML. SGML viewers can take advantage of various styles as well as the header structure to display SGML documents in a manner similar to the printed versions. With SGML authoring or viewing software, users can also browse, search, update and print their documents directly from SGML.

However, from a database perspective, less progress has been made. Currently there are very few SGML-aware database systems. Those that do exist either convert SGML into another database format for the purpose of querying, or create their own proprietary index structures for SGML, allowing some interesting but not by any means complete set of queries. Moreover, the interfaces for these queries are mostly command-line or form-based, and are not intuitive for structured documents. In this section, we report a case study of four database systems for SGML. These cases are selected from two mostly adopted approaches: (a) using a proprietary indexing mechanism, and (b) using existing database technology, such as a relational database or an object-oriented or object-relational database.

3.1 Proprietary Indexes: Open Text and Patricia Trees

Patricia Trees (or *Pat*, in short) were initially applied by Gonnet [Gonnet, 1991] for the purpose of indexing structured documents to perform efficient searches in structured text. This method was applied to a commercial document database by Open Text [Open Text, 1994]. In this product, special indices which can be used for searching the documents very efficiently, are built on top of text. Queries on the documents, however, are limited to ad hoc type queries like finding prefixes of strings (most efficient), some regular expression searches, proximity searches, and combinations of these. Searches are also possible to be delimited by SGML regions. Although basic searches are very efficient, they are not general. Conceptual queries are hard to convert to Open Text's query language, and the conversion process is often not trivial. The supported query language does not form a formally complete set of queries.

3.2 Use of Existing Database Technology

Another significant approach to modeling document databases is the use of an existing database technology. In this approach, the documents are generally converted to database objects during storage time and converted back to the original document form when needed. Because of the involvement of this type of mapping to an incompatible system, all these approaches suffer from lack of closure and also have the possibility of losing information contained in the original SGML document. Section 4.1 gives more details on how information may be lost in the process of transformation. In this section, three current systems using relational, object-oriented and object-relational databases are being considered.

3.2.1 LivePage™ from Inforium

LivePage™ is a recent commercial product which uses relational database technology to store document-based databases. Although relational database technology is the most widely used database technology, it is not powerful enough to model the complex nature of documents, because of their inherently hierarchical nature. Relational databases, on the other hand, are based on tables - which have a very simple flat structure. To model a document database using a relational database, the document needs to be *fragmented*, or broken into pieces, which can fit into flat tables. These pieces are later *joined* together when necessary. However, for a moderately complex hierarchy, this fragmentation becomes very complicated and affects the performance of the database system.

LivePage™ [Inforium, 1995] uses a different type of fragmentation than the theoretical conversion strategy. The strategy adopted in this product is to use a fixed model for any type of document, treating documents as a chain of words and tags. The information gathered from the documents regarding the placement of the words and tags are placed in relational tables, and later reconstructed during query time. The database consists of one main hypertext table containing information on the searchable nodes in the database, such as node name, its parent and sibling position information, and the level in the table of contents. Other tables containing information on specific word occurrences, link information, multimedia object information, and hierarchical information are also extracted from the database and kept in the relational database.

However, this modeling technique can only be used for a limited number of queries involving words and their occurrences in the document. More complex queries involving meta-data are difficult, and frequently impossible to do using this design. This representation method seems ad hoc and does not have any theoretical basis for completeness, and cannot be used for most of the query types discussed in Section 2.1.

3.2.2 VERSO Project at I.N.R.I.A., France

The VERSO Project at INRIA is one of the most prominent works on mechanisms for processing databases largely consisting of text. One of the major approaches taken in this project is the use of an Object Oriented Database Management System (OODBMS) [Abiteboul et al., 1993, 1995; Christophides et al., 1994]. In this approach, an SGML document is mapped into an object-oriented database (O_2). The work involves mapping techniques between SGML documents and their object-oriented design. Some additional information is incorporated and some definitions modified in order to make this conversion [Christophides et al., 1994]. Subsequently, O_2's query language O_2SQL is used for querying the document. This can also be used for database update of the document [Abiteboul, 1995]. The internal indexing mechanism and the query language of O_2 can be used for posing queries on the converted documents.

The primary problem with this approach is that the mapping technique is not one-to-one. This implies that the information contained in the original SGML document cannot be completely retained in the mapped Object-Oriented database schema [see Section 4.1 for details]. Moreover, the output of the system is not *closed*, since the input and output of the query mechanism are not the same. The input to the database is in the form of SGML documents, but the output is in the form of O_2 objects. In order to properly obtain a closed system, one needs to map the O_2 objects back to SGML, which again raises the possibility of losing original information.

3.2.3 Texcel Information Manager

Information Manager from Texcel International [Holst, 1995] is the first commercial information system for SGML documents. This system has a built-in process-control mechanism for authoring, querying, and storing SGML documents in a repository. The repository is enhanced with version-control mechanisms and can be used for cooperative authoring and publishing SGML documents. This system also stores SGML documents in fragments in a commercial object-relational database system (UniSQL), and the document is reconstructed from the fragments

when necessary. The underlying database system is used to formulate queries on these documents using forms-based interfaces. This system also suffers from the lack of closure and completeness.

3.3 Other related work

In addition to the four products or projects described here, a few other systems and theories have been built or proposed. The list-based algebra [Colby, 1992, 1994] has a very strong theoretical basis, but currently do not have efficient implementation. An extension to this work forms the theoretical basis of the current approach. Other related works use the concept of grammars and parse strings based on the grammars to model the structure of documents, and special operations on these parse strings to query the documents [see for example, Gonnet, 1987]. There are a few other commercial as well as public-domain systems for SGML, but they are not included here because of the lack of available information and publications.

4. Shortcomings of the current SGML systems

As indicated in the above discussion, currently available SGML database systems do not meet the standards of products in the relational domain for one or more of the following reasons:

- Lack of a standard querying mechanism or query language, and the use of languages available in the host database system, or proprietary query constructs to perform a limited set of queries.
- Lack of appropriate query interfaces like QBE in the relational domain.
- Necessity for conversion of SGML documents to the appropriate database format -- causes fragmentation, and in may cases, potential loss of information.
- Lack of closure, implying that the input into the system is SGML while the output is database-specific.

The lack of a standard query language and interface implies that the users of text databases have to adjust to the interface of every different product that they use. We place a considerable importance on closure to make the systems scaleable and extendible. The basic idea of closure is that the output from a query can be reused as the input to other functions, giving a greater power to the query processing mechanism. The next section discusses in detail the importance of this fourth problem.

4.1 Why conversion to other database formats is not appropriate

SGML can be described as a meta-language, or a language for expressing other languages. Documents written in SGML need to conform to a particular grammar specified by its *Document Type Declaration (DTD).* The grammar specified by this DTD is closely related to the *Extended Context-Free Grammar (ECFG)*. However, as will be argued later, ECFG is not completely suited to represent SGML documents. The structure represented by the DTD describes a hierarchy of *elements* or *tags* which may have *attributes* and may have a *content model* involving other elements. Simple content models are easy to represent using conventional complex-object database systems or object-oriented database systems. However, SGML documents may have a very complex content model involving multiple alternative structures with differing relative positions between sub-elements. Another important factor that makes things more difficult is the importance of the relative position of various elements in a document, which is very difficult to model in standard databases. The reason behind this is the genericity property of databases, which ignores relative positions of tuples in a table. Moreover, some special features of SGML such as marked sections, LINK, CONCUR, INCLUDE and EXCLUDE make things even more difficult.

Christophides et al. [Christophides et al., 1994] describe methods to incorporate only two of the above incompatibilities, *ordered tuples* and *union of types.* In their implementation using the O_2 object-oriented database, *Ordered tuples* are implemented using a polymorphism with lists, and the *type union* is implemented using a tag that indicates the alternative type that is chosen. Although these are workable solutions, they demonstrate the incompatibility between SGML and object-oriented databases, and working-around is not the best solution to these problems.

Many of the uncommon features of SGML can not be so easily implemented using conventional databases. Some of these features include “marked sections”, CONCUR and LINK. These features cause the parser to infer special structures from the document by suppressing parts of the document. In order to get around this problem, one needs to design an intelligent parser that can get around the SGML standard and keep information intact by not suppressing parts of the document. For example, if a document containing an “IGNORE” marked section is parsed, the contents of the section will be suppressed and will not be parsed. This will result in loss of information when the

document is reconstructed from the database at a subsequent time.

4.2 What we can learn from Relational Databases

Although we have seen [Section 3.2.1] that relational databases can not be used efficiently to model SGML databases, there are many lessons that we can learn from relational databases. In order to build a database system specifically designed for structured documents, most of the features of relational databases need to be incorporated. Some of these features are:

- A strong theoretical basis on which the query language, internal indexing mechanisms and data structures, and optimization techniques are built
- A standard data definition language (DDL) for defining and inserting data
- Structured Query Language (SQL) [ANSI, 1986], a standard data manipulation or query language (DML) for inserting, updating, and querying data
- Query By Example (QBE) [Ullman, 1988, citing Zloof, 1977], an intuitive interface for specifying queries involving multiple tables and complex query types
- Strong closure - inputs and outputs are in form of tables - enabling the concept of views (outputs of queries stored as virtual tables, and can be used as inputs to other queries)

The rest of the paper will investigate methods for approaching a database system with the above properties.

5. Proposed Solution

The above discussion shows clearly that a need exists for a way to appropriately model SGML documents without losing information and, at the same time, to solve queries efficiently. This requires a modeling approach that matches the SGML standard directly without having to convert to another format. This calls for a strong theoretical basis and supporting implementation capabilities, since without a strong theoretical foundation, the results from the system can not be formally verified.

5.1 Theoretical basis

The theory behind the system described here an extension to Colby's work on list-structured data [Colby, 1994]. The original work by Colby used list-structures to represent data in which sequence is important. Hierarchically structured data such as SGML documents were modeled as finite trees. An algebra based on this structure "provided list-oriented functions, such as searching and updating based on pattern match and list position." [p. 692 in Colby, 1994] The queries were represented using patterns and results to queries were based on matches to those patterns in the document. The most important concept in this work was the introduction of structure in the patterns, instead of keeping the patterns based on single dimensional streams. Conceptually, *the structure patterns* introduced in this work represented hierarchical structures with the meta-data at the internal nodes and the text patterns at the leaves.

The patterns are matched against the original document represented as a complex list using an operator *Find*, the semantics of which is to mark the portions of the original tree that match the pattern. The algebra also proposed set operators, comparison operators, and some other complex operators to restructure the trees, replace or delete portions of the structure, and insert nodes into the structure.

5.2 Implementation Approach

The Colby method provides a very strong theoretical basis for specifying queries in complex structured databases, such as SGML documents. However, efficient implementation of the operators is questionable. For the purpose of the current work, a variant of Colby's work is used. This approach uses the parse tree obtained from parsing an SGML document, and generates lists of trees (or forests) as a result of pattern matching.

To ensure that the information contained in the SGML document is not lost in the process of structure creation, the original SGML document is kept intact and the data structure is built on top of the document with references into the original document. The data structure that is built is very similar to parse trees generated from both the catalog information (DTD) and the actual document. The tree structures are *stringed* (i.e., the nodes representing individual elements in the parse tree are bound together in another data structure for efficient access).

The *Find* operation based on navigation of the stringed parse tree is implemented using a very similar technique followed in small accumulator-based microprocessors. Initializing a main accumulator to the root of the parse tree, a set of instructions from a small instruction set acts on the accumulators and changes them based on the instructions. In this case, accumulators are lists of sub-trees containing partially computed results that are refined by means of navigation instructions. The navigation instructions are obtained from a query plan generated from the user que-

ries by an optimization technique. The optimization is very similar to relational databases, adapted for optimizing navigation on parse trees. The optimization heavily depends on the presence of special optional indices on specific tags. Because of the introductory nature of this paper, the details of the optimization are omitted here.

5.3 Query Language

The proposed query language is an extended form of SQL (Structured Query Language) [ANSI, 1986], the universally accepted query language for relational databases. SQL can not be directly used with structured documents, since it is designed specifically for flat tabular structures. Although it works well with flat structures, it does not directly correspond to the types of queries one would like to perform on structured document databases, which have inherently complex structures. For this reason, SQL is augmented with operations that give the users the capability of posing queries specifically suited for structured documents. These extensions involve the ability of imposing structure navigation paths in the queries, expressing a complex output declaration, and incorporating complex data types built at execution time.

The primary purpose of this proposed language [see Sengupta, 1996 for more details] is to provide a high-level language for writing queries that are mostly or fully independent of the internal structure and implementation of the database. Languages such as Standard Document Query Language proposed in the DSSSL standard [ISO 10179, 1995] uses a language derived from Common Lisp for extracting lists of nodes from a document tree. This is a complete programming language with special constructs for extracting node lists. The extended SQL is a much higher level language that will have a minimal requirement of the knowledge of the internal structure of the document from the users, unlike SDQL, and still be able to pose a significantly complete set of queries.

The following are the major extensions to SQL that are proposed. Details on the usage of these operators and example queries based on these operations are described in [Sengupta, 1996].

- **Composition of the '.' operator:** In SQL, to designate a field F for a table alias T, the '.' operator is used - e.g., *T.F* . This works well for a flat structure, since the '.' indirection only needs to be put once. However, in a complex hierarchical structure, the '.' operator needs to be composed, such as *T.F.G* where T is a document type, F is a top-level element in the document type, G is an element in the content of F and so on.

- **The '..' conjunction:** It is difficult to always give explicit path information in queries. In many cases, the path between two elements or between a document type and an element can be uniquely determined by the source and the destination. A '..' operator can be used to specify only the source and destination for such paths. For example, the clause above can be described by *T..G* instead of having to explicitly give the internal path. In case there are multiple paths between a source element type and a destination element type, the query is satisfied if any of the multiple paths satisfies the query.

- **The 'SELECT DTD' clause:** In SQL, the closure of the query is implemented by specifying columns in the 'SELECT' clause that form the resulting table. In case of SGML, the closure is implemented by generating SGML documents as the result of the queries. If only columns are specified in the 'SELECT' clause, a simple DTD incorporating the given columns as elements in the same level is created, and the result conforms to that DTD. However, it is possible to specify a complex output structure using a DTD in the 'SELECT' clause, and specifying the relationship of the elements defined in the DTD with the elements in the original documents.

5.4 Query Interface

The proposed query presentation can be described as "Querying by Templates". This idea has its roots in the graphical language Query By Example (QBE) -- the most widely used form of query interface for relational databases [Ullman, 1988, citing Zloof, 1977]. In QBE, the tables in a relational database are represented as table skeletons on the screen, in which the users type in search texts in appropriate places to indicate the query. In this case, a simple table is used as a template for searching the internal database.

For relational databases, QBE turns out to be an excellent means for posing complex queries using a simple interface. The primary reason behind this is the use of easily visualizable table skeletons for query formulating. Templates for visualizing complex documents are not always easy to visualize. For most types of documents that have a simple conceptual structure, such as poems or dictionaries, however, this appears to work quite well. For example, if the document type is a poem, the template displayed on the screen can be a poem skeleton. The user can specify

his/her query by entering a query string in an appropriate position of that poem template. The result of the query is shown in the same poem template, thus maintaining presentation closure [Figure 2].

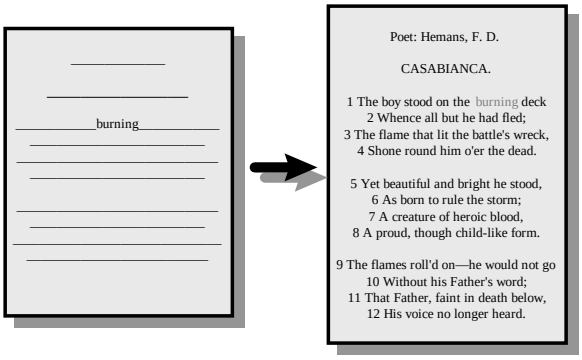

Figure 2. Presentation closure - output is represented in the same template as query input.

For complex queries, multiple instances of a template could be used with the query operations implemented in the same way as QBE. The important difference between QBT (Query By Templates) and QBE is the possibility of structural navigation for structures not visualized at the top level template. For example, in a book template, the only structures visualized in the top level may be *title, author, chapters and lines.* To access other internal structural information such as *publisher, section title,* one needs to navigate into the header or chapter structure. This results in continuous refinement of the template using structural navigation.

The issue of document type invokes the concept of superstructure [Van Dijk et al., 1983], the high level form of narrative information that has been shown to influence users' navigation by affording the development of an accurate model of the information space being explored [Dillon, 1994]. This appears to be a fruitful area for further research, and ties cognitive theories of linguistic processing and human information usage to database design.

Different types of documents are viewed and used in different ways - so having a single interface template for all types of documents is probably not a good idea. A user of a document consisting of poetic works may have a considerably different outlook than a user of a dictionary. However, a generic template that works with any document is also necessary, in case a personalized template is not available. The current implementation emphasizes personalized templates for poems (using the Chadwyck-Healey English Poetry Database [Chadwyck-Healey, 1994]) and a simple generic template for any kind of document.

5.5 Advantages of this approach

Besides the fact that this model is a built-from-scratch model for SGML, it has many other advantages over the current state-of-the-art systems. Primary among them are the following:

- The original SGML document is left intact; indices built on it have pointers into the document.
- It is possible to build indices incrementally, so that changes to the original documents will not result in complete recreation of the indices.
- It is possible to improve performance by using different types of secondary indices like B-Tree indices.
- Query optimization techniques, similar to those in relational database systems, can be applied for improved query performance.
- Closure property is enforced - which implies that the inputs to and outputs from the system are of the same type. This closure is maintained in structure (both the input and output to the database are in form of SGML) as well as in presentation (in the form of the same input and output templates. [Figure 2])

6. A Brief Description of the Prototype System

A prototype system that incorporates most of the proposed enhancements is being developed and tested. At this stage of development, we are placing more emphasis on the functionality and usability aspects of the system rather than search efficiency. Once all the functionality is implemented, the algorithms will be improved for increasing efficiency.

6.1 Platform

The proposed system has a client-server architecture. The server is based on the persistent object storage manager, Shore [Shore, 1994], and requires a UNIX platform with IPC (inter-process communication) and TCP/IP (Transmission Control Protocol / Internet Protocol) support. The major storage requirements and functions are handled by Shore, which stores the SGML files (data) and the indices built thereon as well as the structural information (catalog). The database system consists of a number of clients of the Shore server both for creating the data objects and building the necessary indices, and for handling queries (either using a query language or a graphical query interface).

The query interface is implemented using Java [Java, 1995], an object-oriented distributed programming

language. In spite of its state of infancy and sub-optimal efficiency, we decided to choose Java over other graphical interface builders, because of its availability, and the capability of running Java-based programs from a WWW browser. An alternative search interface using standard WWW forms was also implemented for users without access to a Java-enabled browser.

6.2 Current Status and Availability

The system under construction has two distinguishable components: (i) the front-end interface component and (ii) the back-end query engine component. Both the components are in a post-design, development stage.

An implementation of the QBT interface for poems has been successfully developed using Java, and has been recently put through a preliminary usability test. Performing 9 standard querying tasks, both novice and expert users showed no significant differences between this interface and traditional form-based interfaces in terms of effectiveness and efficiency, but the new interface led to significantly higher satisfaction ratings across both user types [Sengupta and Dillon, 1996]. The results of the usability analysis are being used for improving the design of the interface, and another test of usability is planned after redesign, with the goal being to increase user-effectiveness or efficiency in line with the satisfaction gains obtained. The current version of the interface is available for demonstration on the WWW [see Poetry2, 1996 for the complete URL]. Once again, because of the copyrighted nature of the underlying database, the results of queries will not be available for people accessing outside Indiana University.

The internal search engine is also under active development. Most of the querying constructs are currently in place. For the usability testing of the QBT interface, a major part of the search engine was built using the Pat search engine [Open Text, 1994], but the final implementation will be independent of external database engines.

7. Future Plans

A number of enhancements are planned for the near future after the basic engine and interfaces are completed. These include:

- Development of an online interface using the JAVA™ language
- Implementation of regular expression searches, with at least the features in the SQL “LIKE” clause
- Implementation of queried update, in addition to the file-level update currently used
- Implementation of version-control mechanism, possibly built on top of the current implementation
- Support for HyTime architectural forms [ISO/IEC 10744 described in DeRose, 1994]
- Compatibility with the SQL3 standard when the standard is available [Melton, 1995].

8. Conclusion

This paper attempts to demonstrate the need for an affordable and efficient method for implementing a database system for structured documents. A feasible solution is proposed based on other public-domain components using a unique model for SGML documents and implements a workable version of that model. The basic idea of the system is to put a set of SGML documents in a repository and immediately be able to pose queries on those documents. A query language based on the SQL standard and a query interface based on the QBE interface are also proposed. It is hoped that a complete implementation of this method will have many advantages over even the relational systems in its expressiveness and extendibility, and will definitely be a step towards the databases of the future.

Acknowledgments

We are very thankful to the subjects of our usability analysis for their participation and cooperation. We are also grateful to the LETRS (Library Electronic Text Resource Service) subdivision of Indiana University Library, and especially ex-directors Richard Ellis and Mark Day for allowing us to use the Chadwyck-Healey Database for this research. We would also like to thank Prof. Dirk Van Gucht, Computer Science, Indiana University, for his careful reading and useful comments.

References

[Abiteboul, 1993] Serge Abiteboul, Sophie Cluet, Tova Milo. Querying and Updating the File. *Proceedings, 19th Intl. Conference on Very Large Databases,* 73-84, 1993

[Abiteboul, 1995] Serge Abiteboul, Sophie Cluet, Tova Milo. A Database Interface for File Update. *SIGMOD Record,* 24(2):386-394, June 1995.

[ANSI, 1986] ANSI X3.135-1986. Information Technology - Database Languages - Structured Query Language (SQL). American National Standards Institute. New York, 1992.

[Chadwyck-Healey, 1994] Chadwyck-Healey. The English Poetry Full-Text Database. The works of more than 1,250 poets from 600 to 1900, 1994.

[Christophides,1994] V. Christophides, S. Abiteboul, S. Cluet, M. Scholl. From Structured Documents to Novel Query Facilities. *SIGMOD Record,* 23(2):313-324, June 1994.

[Colby, 1992] Latha S. Colby. An Algebra for List-oriented Applications. *Technical Report 347, Indiana University,* December 1992.

[Colby, 1994] L. Colby, L.V. Saxton, D. Van Gucht. Concepts for modeling and querying list-structured data. *Information Processing and Management,* 30(5): 687-709, 1994.

[DeRose, 1994] Steven J. Derose, David G. Durand. Making Hypermedia Work: A User's Guide to HyTime. Kluwer Academic. 1994.

[Dillon, 1994] A. Dillon. Designing Usable Electronic Text: Ergonomic Aspects of Human Information Usage. Taylor & Francis. 1994.

[Gonnet, 1987] Gaston H. Gonnet, Frank Wm. Tompa. Mind Your Grammar: a New Approach to Modelling Text. *Proceedings: 13th Intl. Conference on Very Large Databases,* 339-346, 1987.

[Gonnet, 1991] Gaston H. Gonnet, Ricardo A. Baeza-Yates. Lexicographical indices for text: Inverted files vs Pat Trees. Technical Report TR-OED-91-01. University of Waterloo, 1991.

[Hammond, 1989] N. Hammond, L. Allinson. Extending Hypertext for Learning: an Investigation of Access and Guidance Tools. *People and Computers V,* 1989.

[Holst, 1995] Sebastian Holst. Database Evolution: the View From Over Here (A Document-centric perspective). *Proceedings of the SGML '95 Conference*, December 1995.

[HTML Working Group, 1995] T. Berners-Lee, D. Connolly. HyperText Markup Language - 2.0. Specification of the HyperText Markup Language from the HTML Working Group. Nov 22, 1995.

[Inforium, 1995] Inforium Inc. Livepage™ A System for Open Information Exchange, *Software Promotion Brochure,* 1995.

[ISO 8879, 1986] ISO 8879. Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML), 1986.

[ISO 10179, 1995] ISO/IEC DIS 10179.2. Document Style Semantics and Specification Language (DSSSL). Working Draft, 1995.

[Java, 1995] Sun Microsystems. Java™: Programming for the Internet. 1995.

[Melton, 1995] Jim Melton. Accommodating SQL3 and ODMG. ANSI Technical Committee X3H2, Database. Document X3H2-95-161/DBL:YOW-32, 15 April, 1995.

[Open Text, 1994] Open Text Corporation. Open Text 5.0 Software and Reference Manuals. 1994.

[Poetry1, 1995] An interface for hypertext browsing or searching the Chadwyck-Healey English Poetry database.WWW URL: http://www.cs.indiana.edu/hyplan/asengupt/poetry/Top/TOP.html

[Poetry2, 1996] The SGML Query Interface Project. WWW URL: http://blesmol.cs.indiana.edu:7890/projects/SGMLQuery

[Sengupta, 1996] Arijit Sengupta. Demand More from Your SGML Database! Bringing SQL under the SGML limelight. *<TAG>, The SGML Newsletter,* 9(4): 1-7, April 1996.

[Sengupta and Dillon, 1996] Arijit Sengupta, Andrew Dillon. Usability analysis for Query By Templates: A new method of querying structured text. Work in progress, May 1996.

[Shore, 1994] M. Carey, D. DeWitt, J. Naughton, M. Solomon, et al. Shoring Up Persistent Applications. *Proceedings of the 1994 ACM SIGMOD Conference*, May 1994.

[Ullman, 1988] Jeffrey D. Ullman. Principles of Database and Knowledge-Based Systems. Vol. 1, Computer Science Press, 1988.

[Van Dijk, 1983] Teun A. van Dijk, Walter Kintsch. Strategies of Discourse Comprehension. New York, Academic Press, 1983.