Object Databases for SGML Document Management

Michael R. Olson
West Publishing Company
mrolson@research.westlaw.com

Byung S. Lee
University of St. Thomas
bslee@stthomas.edu

Abstract

We have investigated the use of an object database as a platform for storing and retrieving Standard Generalized Markup Language (SGML) documents. Qualitative studies convinced us that object databases are a perfect fit for supporting SGML document management. Unfortunately, quantitative benchmark results showed that the particular object database management system (ODBMS) product we used was not capable of supporting large scale SGML applications due to certain defects in its system architecture. The most critical defect was a weak support for location-independent persistent object identifiers. We strongly believe, however, ODBMSs in general are perfect platforms and continue the experiment using another ODBMS product. In this paper, we explain why and how an ODBMS fits well with SGML document management applications, describe how the benchmark experiment was performed and what were the results, and finally present a list of features as a recommendation to those interested in developing or using an ODBMS in support for SGML document management.

1. Introduction

Electronic document management refers to storing, updating, and retrieving documents electronically on computers in place of hard copy papers. This application is critical to a business for harnessing the most important asset -- its information. Document management has been handled traditionally in the context of document imaging and text retrieval. More recent trend, however, is the management of *structured* documents and their work flow.

The structured document management became popular due to wide spread use of Standard Generalized Markup Language (SGML) [3,4,5,6,7] within the publishing industry, improved software technology, and the rapid growth of the Internet. In 1986, SGML was adopted by the International Standards Organization (ISO) [5] as a means to separate content from display characteristics so that documents may be reused on multiple platforms. Soon thereafter, the Department of Defense mandated the use of SGML by its suppliers, resulting in a new software industry to support SGML. Many publishers adopted SGML to utilize these software tools. The Internet introduced SGML to millions of people through its use of an SGML implementation called the Hypertext Markup Language (HTML) [8].

The Internet is a marketplace in which each publisher's competitors are a mouse click away. Publishers must be able to quickly and inexpensively add content to electronic publications, derive new publications from existing ones without replicating data, redesign publications dynamically, provide multiple views of the same documents, allow users to annotate documents through electronic "sticky notes." Besides, publishers must be able to create hypertext which survives updates to Internet and CD-ROM products. Large information providers are faced with packaging and repackaging of terabytes of electronic information in order to keep pace with attractive, ever changing upstart services. Tools that once worked well for print are quickly becoming inadequate for electronic delivery.

The Document Style Semantics and Specification Language (DSSSL) [12] and the Hypermedia/Time-Based Structuring Language (HyTime) [9,10,11] are relatively new standards which complement SGML. These standards address the complex issues of dynamic rendering, multiple views of the same documents, electronic "sticky notes," and creation of robust hypertext. World Wide Web (WWW) developers are using DSSSL and HyTime to extend support on the Internet to all SGML implementations so that on-line publishers are not limited to HTML. However, both DSSSL and HyTime require a hierarchical tree-like model of SGML where each node in the tree knows how to find its parents, siblings, and children. For performance reasons, relational systems attempt to filter hierarchy by storing nested SGML elements as Binary Large Objects (BLOBs). To gain the full benefits of DSSSL and HyTime, BLOBs are not a feasible option.

An object database [1,2] is suitable for representing a hierarchical structure of data. Some object databases also provide persistent object identifiers (OIDs), a similar

Proceedings of The Thirtieth Annual Hawwaii International Conference on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

concept to SGML element IDs. SGML document management can be a perfect application of an object database management system (ODBMS). To verify this, we performed benchmarks on a pilot SGML application built on an ODBMS. The results were not very satisfactory due to architectural problems inherent in the particular ODBMS. Our prototype was not scaleable, in large part due to the lack of persistent OIDs. We desire to share the lessons learned through the project with the software community in this paper.

Our contributions in this paper lie in addressing how an object database fits SGML document management well, presenting representative performance benchmark ideas, and making pertinent feature recommendations to the ODBMS community.

Following this introduction, we first describe the effectiveness measure and the business impact of the technology briefly in Section 2. Then, we give an overview of SGML in Section 3 and a qualitative argument of why an object database fits SGML so well in Section 4. Our quantitative benchmark experiences are described in Section 5. Based on our experience, we present a list of feature recommendations to ODBMS manufacturers and users in Section 6. Summary and further work follow in Section 7.

2. Effectiveness Measure and Business Impact

There are many commercial products [14] that support SGML files with a DBMS. Most of them were using a relational DBMS and as such exhibited limitations in handling documents efficiently -- both in time and storage space. Investigation of using an object database had remained in the research territory [16,17,18]. However, we recently began to see a growing number of commercial products using an object database [14] and believed this should be the general trend on the market. [15] What still remains as an issue is the *scalability*. When millions of documents are stored as objects in an object database, would the system scale up well without being "choked" or experiencing significant performance degradation? This will be the criterion for judging how effective object databases are as an SGML document repository.

In our work, we deal with the scalability issue occurring when object databases are used to load, store, search, and assemble SGML documents. The potential business impact will be tremendous once the technology is proven to be scalable. The market size of document management software has been increasing at the rate of 50% - 60% in 1995 and 1996. [13] A successful result of this on-going research will do nothing but accelerate the explosion of the SGML/DSSSL/HyTime document management systems built on object databases.

3. Overview of SGML

3.1. How Does SGML Work?

There are three parts to an SGML document: (1) the SGML declaration, (2) the document type definition (DTD), and (3) the document instance. The SGML declaration specifies which characters are allowed within the document and indicates the syntax for defining markup delimiters. A given publisher generally uses one SGML declaration for all of its documents. The DTD specifies the structure of a given class of documents. The building blocks of documents are elements, attributes, and entities. DTDs define what element names and entity names are allowed, how often an element may appear, the order in which element names may appear, the types of content allowed within any given element, and the attributes which may be associated with each type of element.

3.1.1. SGML Declaration. "Each SGML document transferred to another system must start with a declaration, called the SGML declaration, which defines the coding scheme (syntax) used in its preparation."[3] Almost every syntactic aspect of SGML can be *customized* within the SGML declaration, which itself is divided into six major sections, as described below:

(1) CHARSET: Specifies the set of characters which are valid within a document.
(2) CAPACITY: Specifies things like the upper bounds on the length of element names.
(3) SCOPE: Specifies the scope (document or sub-document) of the declaration.
(4) SYNTAX: Specifies the rules for defining element names, attributes, and other markup codes. The SYNTAX portion of the SGML declaration is perhaps the most complex, since it has eight sub-categories, some of which have their own sub-categories.
(5) FEATURES: Specifies which optional SGML features are utilized within the document.
(6) APPINFO: Specifies, mainly for human use, application specific information.

3.1.2. Document Type Definition. A DTD is a grammar to which document instances must conform. Each document instance is essentially a large regular expression which may be validated for conformance to its DTD using a parser. An English paraphrase of a specific

DTD might read: 'A *book* is an element which has a title followed by one or more *chapters*. A *chapter* is an element that contains a *chapter-title* followed by one or more *sections*. A *section* is an element which has a *section-title* followed by one or more *paragraphs*. A *paragraph* contains text and may contain *cross references*.' Figure 1 shows how one might write a DTD that fits this description. An **ELEMENT** declaration always follows the pattern "<!ELEMENT name S E contents> where S indicates whether or not the start tag can be omitted and E indicates whether or not the end tag can be omitted." [3] A dash (-) indicates that it is required, while an 'O' indicates that it may be omitted. An **ATTLIST** declaration is used to associate attributes with elements. In our example, we have used two special types of attributes: ID and IDREF. The 'targid' attribute is of type ID. Its purpose is to uniquely identify each paragraph element. The 'targref' attribute is of type IDREF. Its purpose is to establish a logical link to the element with the corresponding ID value.

Note that our simple DTD declares only elements and attributes. A key building block found in most SGML documents is that of an 'entity'. Entities are similar in concept to macros in programming languages. By including a reference to an entity, SGML documents get an in-line copy of the contents of the entity when the document is formatted for print or on-line display.

```
<!DOCTYPE BOOK [ <!-- Note that "<!-- " and " -->" are comment
delimiters -->
<!-- Typically, the first element name matches the DOCTYPE name -->
  <!ELEMENT book  - -  (title, (chapter)+)   +(xref)>

<!-- The "-O" means the chapter element must have a start tag, but that
the end tag is optional. We can do this because the document structure
implies the presence of the end tag, whether it is present or not. In other
words, a chapter may end when the end tag is reached, when the start of
another chapter is found, or when the end tag of the book element is
reached. -->
  <!ELEMENT chapter  - O  (title, (section)+)>

  <!-- The plus sign (+) means the item must occur at least one time
and perhaps many times -->
  <!ELEMENT section      - O  (title, (paragraph)+)>

  <!-- The start and end tags for a title is optional -->
  <!ELEMENT title    O O(#PCDATA)>

  <!ELEMENT paragraph - O  (#PCDATA)>
  <!ATTLIST paragraph  targid ID  #REQUIRED>

  <!-- Both the start and end tags for xref are required. -->
  <!ELEMENT xref    - -  (#PCDATA)>
  <!ATTLIST xref    targref  IDREF  #REQUIRED>
]>
```

Figure 1. Sample DTD based on our paraphrase

3.1.3. Document Instance. We may produce as many documents as we like based on our DTD. In our sample document instance shown in Figure 2, we used some of the tag minimization features provided by SGML. For example, we left out the start and end tags for the title elements and the end tags for most of the other elements. Other document types have different semantics, but each type of document may share the same software tools for viewing, printing, editing, and verifying.

```
<book>This title needs no begin or end tags.
<chapter><title>Here we chose to insert a begin tag without an
end tag.
<section>This time we include only the end <xref
targref=P1>tag</xref>.</title>
<paragraph targid=P1>Notice that the xref may appear
anywhere inside the book element. This is allowed because of
the "+(xref)" inclusion in the content model for the book
element. The "P1" targref matches the "P1" targid, thereby
establishing a logical hypertext link.
<paragraph targid=P2>Also notice that each paragraph is
required to have a unique targid. This is because of the type
specification of "ID" and the "#REQUIRED" in the content
model of the paragraph declaration.</book>
```

Figure 2. Sample document instance based on our DTD

3.2. DSSSL Views Documents as Trees

The purpose of the Document Style Semantics and Specification Language (DSSSL) is to make specifications for the visual rendering of SGML-encoded documents portable between delivery platforms. The DSSSL specification defines two independent processes: (1) the SGML Tree *Transformation* Process (STTP) and (2) the SGML Tree *Formatting* Process (STFP). Both STTP and STFP manipulate SGML documents as tree structures. STTP generates new views of existing SGML documents by moving and reconnecting nodes in a tree. STFP generates a "pretty print", which is a formatted display of SGML for on-line viewing or hard copy delivery.

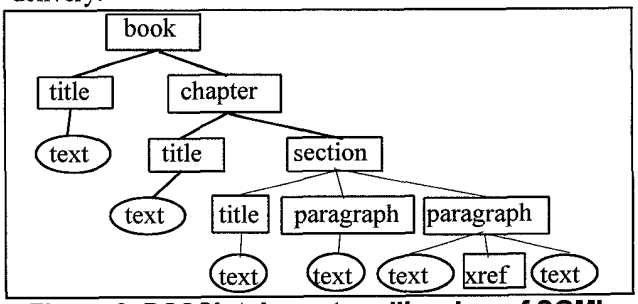

Figure 3. DSSSL takes a tree-like view of SGML

An example is shown in Figure 3. Note that both the STTP and STFP processes would take this sort of logical view of documents, where each element knows its parent and its children. Powerful structural transformations can

be made simply by reconnecting nodes. A "title" with a "chapter" parent can be formatted differently than a title with a "section" parent.

3.3. HyTime Connects Tree Nodes Through Hypertext

The purpose of HyTime is to define a formal means of representing connections between information elements [9]. These connections are generally called *hypertext*. HyTime provides three basic location methods which may be combined as necessary to resolve hypertext relationships: *naming*, *counting*, and *pattern matching*. Naming is the idea of referring to objects via their formal entity name or ID attribute value. Counting has to do with finding an object relative to some other object (e.g., the third item in a list). Pattern matching has to do with finding an object based on a certain set of property values, such as element content or attribute values. It is common to combine these three functions in the following sequence:

1. First, use the name of an entity or the ID of an element to start the search at a node in the tree that contains the target location.
2. Next, use a tree traversal counting method to move down the tree to find a specific descendant of the starting node -- for example, find the node's first child and then get that child's second child.
3. Once you have reached the target element, you may use pattern matching functionality to find a specific range of text -- for example, find the word "Jesus" which is followed by the phrase "is the Son of God". This final step is generally not necessary since most things worth referencing are delimited by SGML element tags.

4. An Object Database Fits SGML Very Well

Management of SGML-encoded documents is a perfect fit for an object database. In particular, the object database concepts of container objects and location-independent, persistent object IDs have potential to decrease the cost of new products through reuse of SGML elements, to increase the power of hypertext, and to allow fully integrated electronic products where SGML elements on CD-ROM know how to find their on-line counterparts. Since both HyTime and DSSSL require a tree-like model of SGML, where each element knows its parent(s) and its children, it is critical that publishers find a way to fully model SGML hierarchy on a large scale.

4.1. Hierarchy

An obvious reason for utilizing an object database to manage SGML-encoded documents is the recursive nature of the data -- elements contain elements which contain elements. Relational databases can model recursive relationships but at the expense of complex, application-dependent code. [1]

In order to minimize the difficulties posed by the hierarchical nature of SGML, relational-based document management systems require SGML documents to be filtered before loading them to the database. The filtering process divides documents into editable binary large objects (BLOBs) of text. Each BLOB may contain several nested levels of SGML elements. So long as filtering decisions are designed well, this method provides adequate performance and allows significant portions of text to be reused in multiple documents. Unfortunately, publishers may find they need to share SGML elements which are "trapped" inside "BLOBs" in the database.

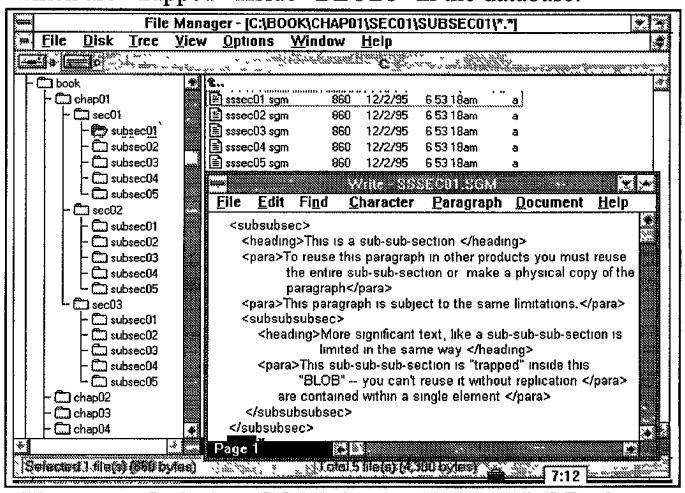

Figure 4. Splitting SGML into editable BLOBs in a relational database

For example in Figure 4, a book has been divided into manageable units. Each type of folder on the left corresponds to a relational table. Each relational table contains information as to which parts are nested inside the particular components. The book relation lists the chapters contained in the book. Chapter relations list sections, sections list sub-sections, etc. Note that, in our example, sub-sub-sub-sec cannot be reused without reusing the sub-sub-sections in which they are contained.

In contrast, an object database can define general classes (e.g., SGMLElement, SGMLAttribute, SGMLEntity) which can accurately model any valid SGML document without filtering the data before loading it to the database. Note that the object database approach reduces human effort in deciding what to filter, and prevents regrets caused by poor choices in what to filter. Perhaps even more exciting is that in the object database

approach, each and every SGML element may be shared by multiple documents.

4.2. Object IDs

SGML and many object databases share the concept of logical identifiers. In SGML, these identifiers are defined by declaring an attribute of type "ID" and specifying in the DTD that the value is required. The values of these attributes are left to the processing application -- they may be inserted by the parser. The IDs inserted by parsers are unique only within the scope of a single document. Mary Loomis points out in [1] that "All denotable objects have a unique immutable identity. Each object has a separate existence and can be distinguished from all other objects. When an object is created, the 'system' assigns it an object identifier, commonly referred to as its *Object_id*."

HyTime concepts combined with object database location-independent identifiers have potential for the creation and delivery of sophisticated hypertext applications. Editors could use full text searches to find targets of hypertext, then the system could maintain bi-directional hypertext through object IDs (OIDs). Inside the database, the OID of the target element alone would be enough to retrace the hypertext inserted by the editor. Outside the database, the OID of the product and the OID of the element would be used together.

Hypertext and electronic sticky notes could follow an object between updates and diverse products (CD-ROM, database, and WWW). (See Figure 5.) CD-ROM products could be republished without concern about how to remap sticky notes -- they would automatically reattach to the elements with the same IDs as in the previous release. Users could create an electronic sticky note on a CD and whenever they come across the same SGML element in an on-line service, the CD sticky note could easily be related to the on-line version. Users could establish hypertext links to an on-line database, and if they purchase CD versions of the targets, the links could automatically be redirected to their CD-ROM publications.

Figure 6 shows how one might utilize HyTime's *clink* concept to carry object IDs outside a database onto CD-ROM and other delivery platforms. We assume that the delivery system maintains a catalog of books (in the world of SGML a book might be called a "document entity") available to the user. The catalog (sometimes called an "entity map") might indicate that the target book is located on a particular CD-ROM. An SGML viewer could find the oidTargBook value in the catalog, instruct the user as to which CD to insert into the CD-ROM drive, and then jump to the appropriate element on the CD (the one corresponding to the oidTargElement value).

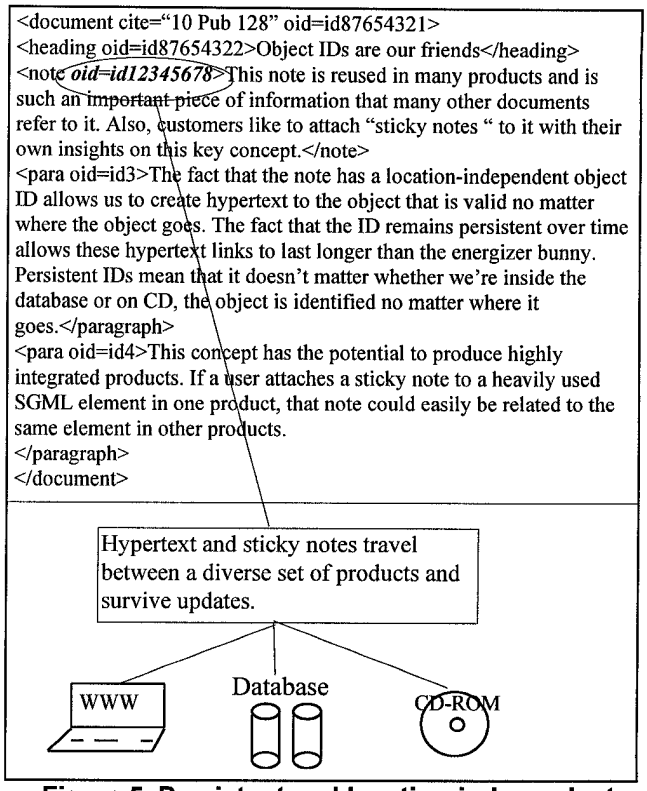

Figure 5. Persistent and location-independent IDs help with product integration

```
<!ELEMENT   citation - -  (#PCDATA)>
<!ATTLIST   citation
  oid               ID        #REQUIRED
  oidTargBook       ENTITY    #IMPLIED
  oidTargElement    NAME      #IMPLIED
  HyTime            NAME      "clink">
```

Figure 6. HyTime's clink allows us to use OIDs outside the database

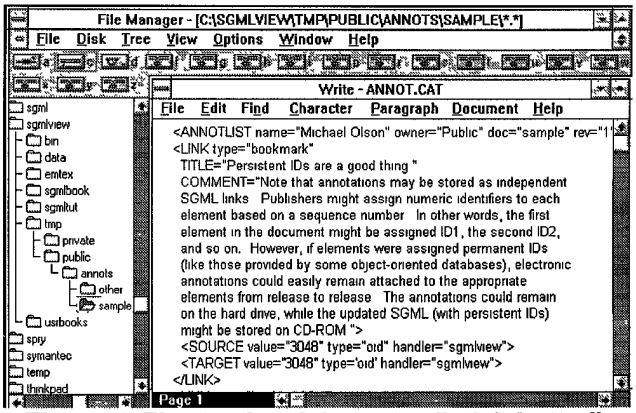

Figure 7. Electronic sticky notes "stick better" with persistent IDs

Figure 7 shows an example of an electronic sticky note as it is stored on a user's hard drive by a commercial SGML viewer. Let us use Figure 7 to discuss a problem

with the design which can be easily remedied through persistent IDs. The "LINK" element may be utilized as a book mark, or as an annotation (i.e., electronic sticky note), or even as a hypertext link between "SOURCE" and "TARGET" elements. The problem with a commercial design is that the "value" attribute is based on temporary IDs that are assigned to each and every SGML element as the electronic book is indexed. The next time the book is published, the electronic annotations will be invalid because the IDs will be different in the new release. Persistent IDs assigned to each and every SGML element eliminates the problem. Generating persistent IDs, however, is a problem in itself. Implementing distributed database systems which can retrieve information based on IDs is another problem. Retrieving the nested parts of a publication from a database for publication on CD-ROM is a third problem, particularly for relational systems. (See Section 4.1.) Some object databases solve these problems automatically.

4.3. Semantics

Rendering or constructing new products are facilitated by knowing the structure of the document. Parents must know their children, children must know their parents. References must know their targets, targets must know who references them. The semantics of SGML allows for "nested fields" so that one should be able to find a large object by first finding an object it contains.

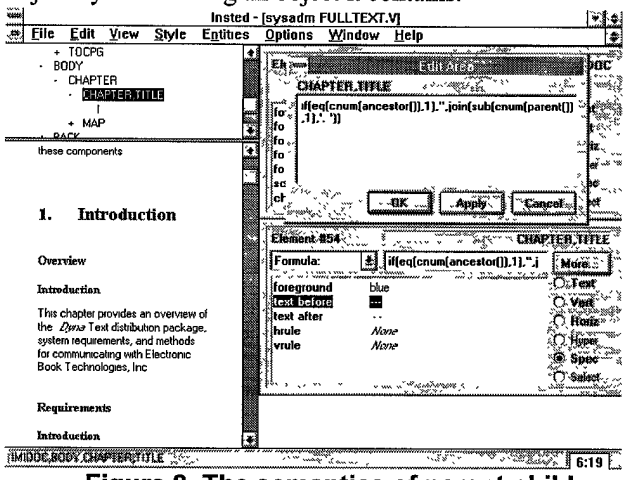

Figure 8. The semantics of parent-child relationships facilitates rendering

Figure 8 shows an example from a commercial style sheet editor. It illustrates the power of parent-child relationships in rendering SGML documents for display. Each element's context (its parent-child relationships) is used by the editor in defining rendering characteristics for the element. The style sheet editor allows text to be propagated from one or more ancestors or from child elements. Chapter numbers can be assigned dynamically depending on the relative location of the chapter elements. The same techniques used for dynamic rendering on CD-ROM could be used in an authoring environment. It could appear to users that section headings are propagated to their respective sub-sections, without actually replicating data.

An SGML document management system could be designed to support the API's necessary to cooperate with a style sheet editor and an SGML viewer. As DSSSL evolves, object databases could be designed to respond directly to the SGML Tree Transformation Process (STTP) and (2) the SGML Tree Formatting Process (STFP) functions. Doing so would allow multiple vendors to develop SGML viewers and editors that can operate on any "DSSSL-compliant" database. Document management standards have not yet reached this level, however it seems a natural extension to DSSSL, and many document management vendors are talking about ways to standardize their API's.

An SGML structure editor could benefit from an object database approach to SGML storage. Mary Loomis points out that "A text editor must encode a document's structural information explicitly in the document. Reading or modifying the document requires the application to decode the document's structure. By contrast, an object database could store the document as a container object, and the object DBMS could understand the structure of that container. A document is an object composed of other objects. An application can request access to a particular component object, say the list of references or a specified chapter, without having to decode markers in the bits stored as a document BLOB."[1]

4.4. Distribution

In a relational system, distribution of data is done through one of three methods: (1) certain relations are partitioned horizontally based on ranges of values in one or more attributes; (2) relations are partitioned vertically so that some columns reside on one server and some on another; or (3) a combination of (1) and (2). These methods are useful in many cases since they keep pockets of data relatively small for backup, recovery, and performance reasons while giving end users the illusion of a centralized database. For example, one might have a class of reusable SGML components called "pages". Each page might correspond to a specific publication, and the relational table which holds the pages might be split horizontally based on "publication number" as illustrated in Figure 9.

User Query:

```
SELECT pagetext
FROM pages
WHERE pagenum=30 AND pubnumber=523
```

The DBMS analyzes the query and searches the appropriate table based on "pubnumber" value.

SERVER 1: (PUBNUMBER 1 through 499)

PUBNUMBER	PAGENUM	PAGETEXT
001	0001	Text
001	0002	Text
...	...	...
499	0001	Text
499	0002	Text

SERVER 2: (PUBNUMBER 500 through 999)

PUBNUMBER	PAGENUM	PAGETEXT
500	0001	Text
500	0002	Text
...	...	...
999	0001	Text
999	0002	Text

Figure 9. Example of horizontal partition method of a relational database

In an object database, distribution can be made on a much more ad hoc basis. Objects might be composed of other objects, each residing on different servers, and retrieval of objects and their parts can be done transparently to the application. Note that distributed databases are not merely a buzz word. They are a necessity for the usual terabyte needs of publishing industry. Even mainframe systems are unable to store all documents in a single relational table. (The maximum relational table size in some large scale mainframe systems is 64 gigabytes.) It seems reasonable to assume that object databases face similar limitations, but object databases with location-independent object IDs have the potential to succeed by using the divide-and-conquer approach.

5. Benchmark Experience

While the object database paradigm offers a very elegant means for modeling SGML, it is a relatively new technology and there may be difficulties in finding an ODBMS that can scale to our needs. In our opinion, the ODBMS used in the project (which we will call "ODBMS-X" for the rest of this paper), was not well suited to our needs. While performance when retrieving heavily nested SGML is quite good, the system is unable to manage databases larger than the maximum file size supported by the client machine's operating system. In general, this means that 32 bit client machines are limited to accessing 4 gigabyte databases. Since ODBMS-X reserves some addressing bytes for its own use, even on a 32 bit machine it is a non-trivial matter to create a database larger than 200 megabytes.

ODBMS-X used in the project provides a feature called an "object reference", which we thought was a location-independent ID that remained static throughout the life of the object no matter where it is moved. We planned on using object references heavily in order to create a scaleable prototype. However, the ODBMS-X's references are not persistent. If you move an object, the reference changes. ODBMS-X attempts to automatically remap relationships defined by object references whenever related objects are moved. This may maintain data integrity inside the DBMS, but it precludes our use of object references outside the database (say, for example, as SGML element ID values).

5.1. The benchmark prototype

The prototype is composed of five modules -- a Load Utility, Archive Utility, Query Engine, Product Generator, and Hypertext Engine. The *Load Utility* is used to load the content of an SGML-encoded ASCII stream into a database by creating objects. The Load Utility recognizes the basic building blocks of SGML (elements, attributes, and entities). The *Archive Utility* is used to dump the content of persistent database objects to an SGML-encoded ASCII stream. The Archive Utility relies on each object to "print itself" so that the utility can merely act as a driver.

The *Query Engine* forms the basis for searching the database and finding objects to be collected into a new product. The Query Engine could potentially be used in either interactive or batch mode to establish hypertext relationships between elements. See Figure 10 below for an overview of the syntax and an example of the use of our query language.

```
                /ELMT(val1 val2 ... valn)   [vals are OR'd]
               / ATTR(val1 val2 ... valn)   [vals are OR'd]
AND'd <
               \ WORD(val1 val2 ... valn)   [vals are OR'd]
                \ENTY(val1 val2 ... valn)   [vals are OR'd]
Example:
  "ELMT(TITLE) WORD(INSURANCE) WORD(AUTO CAR)"
```

Figure 10. Query language syntax and example

The *Product Generator* was implemented utilizing the Query Engine. The lack of persistent OIDs prevented us from implementing the *Hypertext Engine.*

5.2. The benchmark results

Scale: ODBMS-X works well for databases under 200 megabytes. If developers are willing to learn special techniques for larger databases, it may even be possible to create a database as large as 4 gigabytes on a 32 bit operating system. However, even the ODBMS-X manufacturer admits that larger volumes of information must be split across databases. SGML is purely hierarchical in nature and finding ways to ensure that document instances never exceed the 200 megabyte limit prohibits our use of a recursive object model. When one considers that word indexes can be equal in size to the source data, the 200 megabyte limit makes the product virtually unusable.

Load Speed: One goal was to load and fully index 500 megabytes of source SGML in 48 hours. The fastest load rate we achieved was 1,435 bytes per second when loading 22 megabytes. At this rate it would take approximately 100 hours to load 500 megabytes. As we made modifications to improve the expansion ratio explained later, load performance dropped further to 507 bytes per second. These results were not promising, although the problem is probably in our implementation rather than in the ODBMS. The main flaw in our implementation has to do with how a "locator dictionary" was implemented. We had some difficulty in utilizing ODBMS-X's indexes and query language and hence created our own *linear* search algorithm. The larger the number of entries in a locator dictionary, the slower the lookup on words. Had ODBMS-X shown more promise with regard to scalability, we would have spent more time attempting to resolve the indexing issues.

Expansion Ratio: The expansion ratio (the size of the fully indexed database compared to the size of the source data) was also an issue for the prototype. The original configuration expanded at a rate of 9 to 1. Most of the object classes in the prototype have at least one "container" (a data structure that holds pointers to other objects). By being stingy on the initial size of these containers, a considerable amount of space was saved. By reducing the expected number of owners of an element from 2 down to 1, the expansion ratio dropped to 6.7 to 1. The best expansion ratio number was 3.8 to 1 for the most recent configuration. (In order to achieve this value, the initial size of the number of expected locator pointers was reduced from 1000 to 10. Besides, the percentage of unique words and SGML attributes was increased by a factor of three so as to reduce the average number of elements pointed to by any given locator. As stated in the "Load Speed" section, these changes hurt load performance.) Once again, had we used ODBMS-X's index capabilities and its query language, load performance would not have been an issue.

Querying (Cold and Warm): Query performance was within the bounds of original project goals. For small databases consisting of a few megabytes of source, two to three term queries completed in 20 seconds on average for "warm" queries and 30 seconds for "cold" queries. Larger databases of 20 or more megabytes required 48 seconds on average for "warm" multiword queries and 98 seconds for "cold" ones.

Note that had we used ODBMS-X's indexing capabilities and its query language, performance would have been better. However, full text indexing would likely be carried out utilizing third party software and without the aid of the ODBMS -- third party full text indexing and search packages offer superior functionality and performance. Note that because ODBMS-X does not include the concept of persistent, location-independent logical object IDs, incorporating third party full text indexing would be difficult with this ODBMS.

Export Speed: The export speed (the rate at which SGML can be copied from the database to a flat file) is the brightest spot in the benchmark. When relational systems attempt to model the full SGML hierarchy, for example by modeling each SGML element as a tuple in a relational table and each level in the hierarchy as a separate table, the retrieval performance becomes unacceptable.

It was the goal of this project to model the full hierarchy of SGML and still allow 500 megabytes of source SGML to be extracted in a 24 hour period. The fastest export rate for a large database was 21,657 bytes per second; the slowest export rate was 12,807 bytes per second. If these rates hold constant without regard to the size of source, then 500 megabytes of SGML can be exported in 6.4 to 10.8 hours. The project was carried out on a relatively low scale UNIX workstation. One might expect even better performance on a more powerful UNIX server.

The wide range in export performance should not come as a surprise. The slowest export rate was for heavily nested, small SGML elements (as high as 10 levels of nesting). The fastest rate was achieved for rather large SGML documents (each around 32 kilobytes) with relatively little nesting of elements (three levels of nesting on average). The surprising aspect of the variance in performance is that it is so small. Our hope is that a more scaleable ODBMS can be found which provides similar levels of export performance.

6. Recommendation to ODBMS Manufacturers and Users

6.1. Location-independent persistent object ID is the key to scalability

ODBMS-X, the ODBMS used in this project, does not seem well suited to our needs. Performance when retrieving heavily nested SGML is quite good without having to artificially segment the data into "BLOBs". However, the product is not suitable for large applications. It is difficult to estimate how many SGML elements 500 gigabytes of source might translate into, but it is clear that an object database that is limited to 4 gigabytes per database would pose difficulties for data management. It is not merely the fact that 4 gigabytes is a small storage capacity. The problem also involves the inherent weakness in maintaining inter-object relationships without persistent keys (i.e., IDs).

Large volumes of data require periodic reorganization. As long as inter-object relationships are maintained independent of the physical location of objects, reorganization of data does no harm. However, without persistent keys to objects, any change or movement of data can destroy the inter-object relationships, or at least force the use of expensive remapping of the relationships.

If a paragraph holds an average of 500 bytes, and each paragraph holds an average of four sentences, then one might say that a paragraph averages five SGML elements. Assume for a moment that large publishing companies average 500 gigabytes of SGML data. Let's do a quick calculation based on our assumption: (500,000,000,000 bytes / 1000 bytes per paragraph) = 500 million paragraphs, which translates into 2.5 billion SGML elements. If we were to use any database that uses direct pointers to the physical location of objects, it is clear that we risk a situation in which moving a single object forces the remapping of 2.5 billion inter-object relationships. True, the chance that any one element might be referenced by all the other elements is small, but the scenario drives home an important weakness of this approach.

6.2. Additional recommended features

Once the lack of persistent, location-independent object identifiers presented itself as an issue, we started investigating other ODBMS products. Our main criteria in evaluating ODBMS technical documents were whether or not the DBMS provides object IDs that remain persistent over time, are unique across a network of databases, and may be used to find an object anywhere in the network in a manner that is essentially transparent to the application program. We have found a couple of products that have the greatest potential for scale and ease of use. Based on our study, we list here some of the features as a recommendation for ODBMS manufacturers and users.

1. An ODBMS should allow users to create an index on an *arbitrary* collection of objects so that you can organize products by collecting pointers to objects. To improve search performance, it would be nice to be able to index the attributes of objects that are stored in a particular collection only. Unfortunately, most commercial ODBMSs currently force users to create an index on *all* objects of a class, no matter which collection they reside in.
2. Versioned objects should be movable from their birth database to another database residing on the same or different site. In addition, configuration management of versioned objects should be available to keep track of which version was delivered with which release of a product.
3. An ODBMS should support group reads and writes for distributed objects without specifying individual databases. Performance may be an issue if the document management system client must invoke individual reads in order to retrieve a set of objects. If possible, most reads should be done as group reads followed by sequential reads to get the more distributed parts.
4. An ODBMS should use an efficient object search algorithm whose speed is insensitive to the increase in the number of networked servers. Once an object is not found on the connected server, it would be a poor approach to do an exhaustive search on all the other servers. You would want to set up search hierarchies based on criteria such as machines, users, or processes in an attempt to tune the search to the specific situation. The search algorithm should ensure that each node in a network can operate independently of other nodes.
5. Users should be able to add database volumes incrementally and randomly at will. Volumes may consist of raw disk partitions. A single database should potentially accommodate unlimited volumes.
6. Multiple, concurrent, parallel subprocesses are a definite advantage. This creates the potential for executing many behind-the-scene lookups, etc., that can take advantage of a multiple CPU server and/or multi-server environment.
7. An ODBMS should manage objects replicated on different network nodes transparently to users -- synchronously by default and asynchronously as an option.
8. An ODBMS should provide resiliency to a network failure. That is, nodes in a network of database servers can operate even with all network communications are down and without the risk of duplicating object IDs.

Proceedings of The Thirtieth Annual Hawwaii International Conference on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

9. Dynamic reclustering of objects would be an attractive feature because in the publishing industry it is difficult to pre-determine the clustering strategy that maximizes file I/O performance. Data access patterns change over time, and data become fragmented as new products are constructed from existing products. If an ODBMS maintains a statistical trace of data access patterns, then it can utilize the information to recluster objects on a regular basis or triggered by the "event" of reaching a statistical threshold.
10. It will be nice to have an event notification mechanism that will allow users to monitor specific objects and/or all objects of a class and receive a message when an object is created, modified, or deleted. You can optionally specify a predicate to filter events based upon object attribute values. You can also define your own events and receive a message when it occurs.

7. Summary and Further Work

An object database suites SGML very well, at least on a small scale. It remains to be seen how well it manages terabyte needs. With Internet users clamoring for more information, higher quality, and lower costs, and with governments and publishers adopting SGML (and the powerful software tools that come with it) at a rapid pace, it now seems SGML is a requirement rather than an option for the publishing industry. Since HyTime and DSSSL are expected to be the predominant means by which to specify hypertext relationships and the display characteristics of SGML, it seems in a publisher's best interest to find a way to model full SGML hierarchy on a large scale.

The results of our benchmark were not gratifying, particularly in terms of the limited database size and the lack of persistent, location-independent unique object IDs. If ODBMS vendors follow the recommendations in Section 6, their products will be better able to meet the needs of the publishing industry. Most importantly, the effectiveness of using object databases for SGML document management relies on the scalability. Once the current or future ODBMSs meet the scalability need, they will bring a significant impact on the document management market.

We are continuing our study using other ODBMS products while improving and extending the initial benchmark prototype. At the same time, we are investigating relevant technical issues, such as developing a more efficient object search algorithm in a distributed network environment, as a by-product of the study.

References

[1] Mary E. S. Loomis, *Object Databases: The Essentials*, Addison Wesley Publishing Company, 1996.

[2] Won Kim, *Introduction to Object-Oriented Databases*, The MIT Press, 1990.

[3] Martin Bryan, *SGML: An Author's Guide to the Standard Generalized Markup Language*, Addison-Wesley Publishing Company, 1995.

[4] Charles F. Goldfarb, *The SGML Handbook*, Clarendon Press, Oxfird, 1990.

[5] ISO 8879: 1986. *Information Processing -- Text and Office System -- Standard Generalized Markup Language (SGML)*, October 15, 1986.

[6] Eric van Herwijinen, *Practical SGML (2nd ed.)*, The Netherlands: Kluwer Academic Publishing, 1994.

[7] Steven R. Newcomb, "SGML Architectures: Implications and Opportunities for Industry" in *TAG*, pp. 1 - 5, August 1995.

[8] Ian S. Graham, *The HTML Sourcebook: A Complete Guide to HTML 3.0*, Wiley Computer Publishing, New York, 1996.

[9] Steven J. DeRose and David G. Durand, *Making Hypermedia Work*, The Netherlands: Kluwer Academic Publishers, 1994.

[10] Ralph Ferris, and Victoria T. Newcomb (ed.), *HyTime Application Development Guide*, ftp://ftp.techno.com/pub/HyTime/Application_Development_Guide/, February 1996.

[11] Charles F. Goldfarb, *Catalog of HyTime Architectural Forms and HyTime SGML Specification Version 2.0*, http://www.sgmlopen.org/sgml/docs/library/archform.htm, June 28, 1993.

[12] James Clark, *ISO/IEC 10179:1996 Document Style Semantics and Specification Language (DSSSL)*, http://www.jclark.com/dsssl/.

[13] PR Newswire, *Delphi Market Research Finds Document Management Market Shift*, June 3, 1996.

[14] Bob DuCharme, *DBMS Support of SGML Files*, http://cs.nyu.edu/cs_alumni/duchar96/sgmldbms.html, July 12, 1996.

[15] Jim Donahue, *Document Objects*, presented at the Database Seminar, Department of Computer Science, Stanford University, May 10, 1996.

[16] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl, "From structured documents to novel query facilities" in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 313 - 314, Minneapolis, MN, May 1994.

[17] T. Yan and J. Annevelink, "Integrating a structured-text retrieval system with an object-oriented database system" in Proceedings of the Twentieth International Conference on Very Large Data Bases, Santiago, Chile, September 1994.

[18] M. Yoshikawa, O. Ichikawa, and S. Uemura, "Amalgamating SGML Documents and Databases" in *Proceedings of the International Conference on Extending Database Technology*, Avignon, France, March 1996.