Adding Semantics to SGML Databases

Subhasish Mazumdar[1], Gary Yuan[1], Weifeng Bao[1], and Jonathan Price[2]

[1] Department of Computer Science
[2] Technical Communication Program
New Mexico Institute of Mining and Technology
Socorro, NM 87801. USA.

Abstract. Huge collections of linked documents can now be efficiently stored. However, full online access and electronic publishing through reuse of document parts require sophistication and precision in queries. Such a query facility is only possible through the inclusion of appropriate semantic information. Manually adding such information to multigigabyte document sources is daunting for technical writers. Our approach aims at making this task feasible by exploiting a conceptual schema of the enterprise. The result is an integrated schema — one that covers the traditional information system of the enterprise as well as the information that exists solely in the world of documents.

1 Introduction

We can now store electronically every document ever published by an organization, including data sheets, manuals, memos, and reports, and so we can make them available for online access. Hypertext has made it possible to *link* these documents and modern Database Management Systems (DBMSs) have enabled their efficient storage and retrieval in parts or as a whole; sound bites, images, and video clips can now supplement text. These advances have raised the expectation that an online search for information will extract documents and parts thereof with pinpoint precision, but this expectation has not been realized.

The notion of *document* itself has begun to degenerate because we can reuse the material in bits and pieces. Writers are expected to create a new "document" quickly by selecting pieces of existing ones and stitching them together. However, writers attempting such reuse have discovered that a loose, intuitive organization, tolerable in books, causes major problems in the online world [14].

A step in the right direction towards organization of document collections was taken through the adoption of SGML (Standard Generalized Markup Language) [8, 16], as an international standard [11] for document representation. SGML lets us capture the structure of a class of documents (e.g., memos) by defining their *type* through a *DTD (Document Type Definition)* containing precise grammar rules about how and where the logical components (e.g., sections, paragraphs, overviews) can and should occur. All legal document *instances* (e.g., correctly structured memos) must conform to that type.

SGML allows a separation of the document *structure* from its *presentation* promoting document reuse and interchange. The power of word search provided

by traditional Information Retrieval Systems is now augmented by the ability to restrict the search to specified SGML components. In order to get the procedure for painting hulls of ships of class *Container*, we can pose the query, *Return all procedures in the manuals on ships of class "Container" that contain the words 'Painting' and 'Hull'.*

Unfortunately, even this query capability may not be precise enough. First, consider a case where a procedure on painting a hull is about painting its components and the word *hull* does not appear inside the procedure — it appears at a higher structural level such as the enclosing section. Such a situation occurs frequently in books, because the writers assume the reader has moved through the chapter title, chapter introduction, title for the group of procedures, and introduction to the group before actually reading the individual procedure; hence the writers see no need to repeat an "obvious" topic, such as *hull.* Second, consider the case when ship documentation is not organized by classes of ships but by individual ships. In this case, the query needs to replace *ships of class "Container"* with a list of the specific ship names that belong to the class. This list is hard to locate among documents but can easily be found elsewhere: the shipyard's corporate database. Third, consider a rather subtle case when the hull's propeller needs special attention while painting and this fact is mentioned only in the segment of the manual dealing with propellers. This crucial fact would *not* be returned and the consequences could be costly. We will return to this example later. In all these three cases, powerful word search engines are of little use; consequently, the document databases are unable to extract the relevant information with precision.

What is lacking is a fuller articulation of *what the document pieces are about*, i.e., the associated semantics. In addition to the SGML capability, we need to pose and answer the query, *Return information pertaining to painting of hulls of ships of class "Container"*, which is no longer restricted to individual words and structural elements, but relies more strongly on semantic notions such as *painting* (a process), *hull* (a ship component), and *Container* (a product line).

How can appropriate semantics be included? One possibility is to extend the DTD by adding new structural elements reflecting semantic categories. For example, a writer may take the structural element *procedure* and create new elements *procedure for the hull* and *procedure for the cabin.* This causes several problems. First, the option of making ad-hoc changes to the DTD is not welcomed by writers because the process is so difficult and error-prone. Also, such a strategy raises the question whether or not such a change is benign to older documents. Further, the proliferation of elements becomes counterproductive because writers can no longer remember their intent. We therefore need a way of adding semantic information without unbounded changes in the DTD.

Any scheme for adding semantic information must face two problems. First, it is simply infeasible for writers to add lists of semantic notions for *every* structural unit (such as paragraphs) of huge documents. Second, unless there is some correspondence between semantic notions that are likely to be queried and those that writers feel are appropriate, this becomes a futile exercise. Therefore, we

need a methodology for adding semantic information to documents that addresses these concerns; only then can we achieve a powerful document query facility.

In this paper, we outline such a methodology. In the process, we achieve a happier result: an integrated conceptual schema for the entire company. While our approach is especially suitable for medium to large-sized companies, small units following our approach will get increasing benefits as they grow.

In summary, our approach consists of the following six steps:

- The given DTD is modified to allow semantic information to be added.
- From the DTD, an object-oriented document database is created and all existing documents are inserted into it; we refer to this database as DOCDB.
- We work with an integrated conceptual schema $\mathcal{S}$ containing two parts: $\mathcal{C}$, covering the traditional activity of the corporation or enterprise and $\mathcal{D}$, the world of documents, i.e., $\mathcal{S} = (\mathcal{C}, \mathcal{D})$. If the corporate database, to which we refer as CORPDB, exists, it conforms to $\mathcal{C}$.
- Writers study $\mathcal{S}$ and add semantics to document components in DOCDB.
- $\mathcal{C}$ and $\mathcal{D}$ are modified to track changes in the enterprise and in documents.
- Users query the document database using $\mathcal{S}$, i.e., both $\mathcal{C}$ and $\mathcal{D}$.

The paper is structured as follows. We discuss the above steps in the following six sections. Next, we review related work, then state our general conclusions.

2 Modification of the DTD

In order to add semantic information to document components, the DTD needs to be modified. Seven "semantic" attributes are automatically added to each component: $creation_date (the date of creation of the element), $modification_date (date of last modification), $author (a reference to a person in the enterprise), $owner (a reference to a department), $status (the current status of this document element, e.g., *draft, ready, obsolete, confidential, final*), $comments (notes by the author to other authors), and $subject (a list of one or more elements of the schema $\mathcal{S}$). These attributes are made optional so that the changed DTD will accommodate all existing unmarked older documents. Also, it is easy to take our marked document, strip it of these semantic attributes and ship it to other enterprises who use the original DTD.

While the first three of the attributes can be filled in by the system automatically, $owner may have to be occasionally overridden by the writer. Often, the DTD will contain attributes very similar to $creation_date, $modification_date, $status, or $comments; such redundancy results in a waste of space but results in a simpler implementation with no loss in query power.

The last attribute, $subject, is the crucial one for us. It is a list of references to elements in the schema $\mathcal{S}$ and to actual database instances; such semantic information allows a user to query a concept or fact based on $\mathcal{S}$ and find relevant document components.

3 Conceptual Schema

A *conceptual schema* is a (software-independent) high-level description of the structure of the enterprise (or its working database). A *conceptual model* is a language used to specify this schema — typically making use of the abstraction mechanisms *classification, aggregation,* and *generalization.* The Entity-Relationship (ER) model has emerged as the leading conceptual model [2]. It uses the concepts of *entity* (a class of real world objects), *relationships* (aggregation of one or more entities), and *attributes* (elementary properties of entities or relationships). Recently, it has been augmented with generalization hierarchies and composite attributes, forming the Extended ER (EER) model [2].

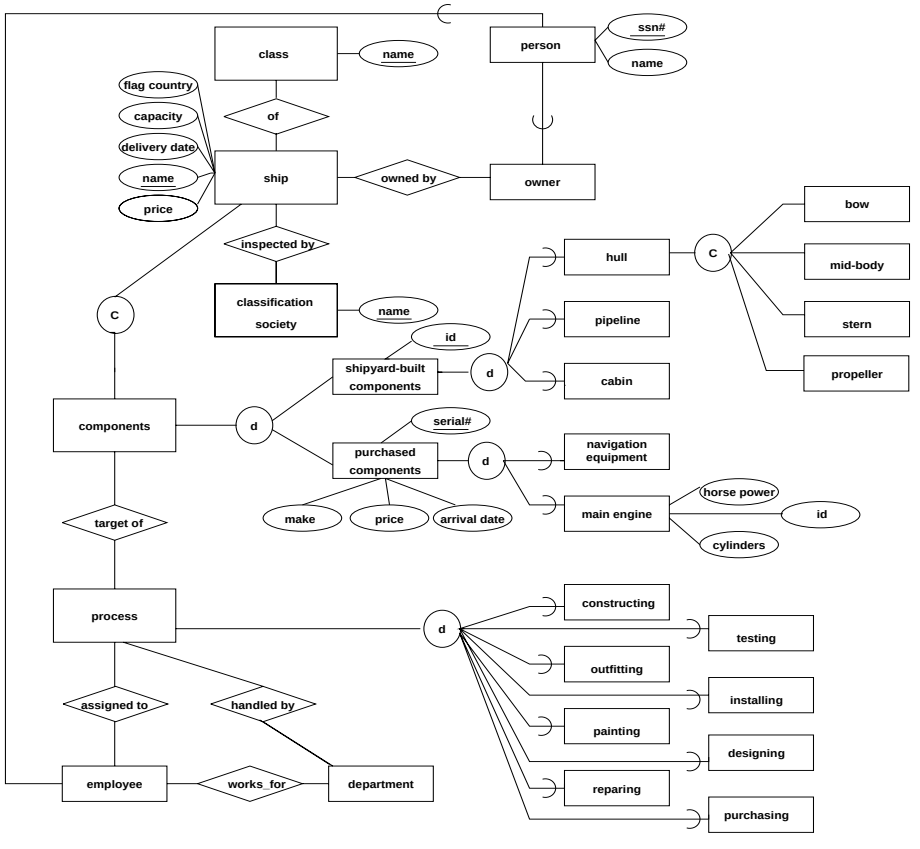

Fig. 1. A fragment of a shipyard EER model. Each instance of the entity *ship* has five attributes of which *name* serves as a unique identifier; the relationship *of* indicates that each ship is of a certain class; the set of *employees* is a subset of the set of *persons*; the set of *shipyard-built components* have disjoint subsets *hull, pipeline*, and *cabin*; each *hull* has four important components, *bow, mid-body, stern*, and *propeller.*

For example, Figure 1 shows a fragment of an EER schema of a shipyard manufacturing ships. Entities are denoted by rectangles with attributes in ovals; unique ones are underlined. Diamonds denote relationships between/among entities. A circled-'d' indicates disjointness of specialized classes, while a circled-'o' indicates overlap. A circled-'c' denotes the relationship *is-component-of*, which is very useful in engineering applications and is part of certain object-oriented models, e.g., ORION [12].

We study the company and its operations and look for a conceptual schema $\mathcal{C}$ to capture it; we will assume without loss of generality that $\mathcal{C}$ will use an EER model. Typically, we find the use of a DBMS for routine transactions for which a high-level conceptual schema may well be available. Otherwise, we construct one by reverse engineering [2].

There may be situations when a document component describes an object, concept, or activity that has not been described in $\mathcal{S}$. The writer then requests the DBA (DataBase Administrator) [1] to extend $\mathcal{S}$ through the addition of one or more elements so that writers in future may use them as \$subject entries. When the DBA grants the request, these elements get added to $\mathcal{S}$ within its $\mathcal{D}$-part. In fact, $\mathcal{S}$ follows an exclusively writer-initiated development having only a $\mathcal{D}$-part if $\mathcal{C}$ is initially empty, as would be the case if the reverse engineering mentioned earlier is impossible.

4 The Document Database

The modified DTD is translated into an object-oriented schema and implemented as an object-oriented database on an object-oriented database management system (OODBMS). We refer to this database as DOCDB, the document database. A method for performing this step appears in [5]. Basically, a class is defined for every structural element with an instance variable for each of its attributes. In order to capture various directives of the DTD, e.g., the title is mandatory but the acknowledgment is optional, some integrity constraints are added.

Once DOCDB has been created and existing SGML documents are inserted into it by storing each component as an object in its appropriate class, the writer starts interacting with the OODBMS using its sophisticated tools. From this point onwards, each SGML document becomes a *virtual* one since it is more efficient to generate it by assembling its components; also, it is then guaranteed to conform to the DTD.

Figure 2 shows a part of the architecture of DOCDB: the instance table and index help speed up search. The instance table contains entity and/or attribute instances used as \$subject entries. The index contains maps of the form $x \rightarrow d$, where x is an entry in an instance table or an element of $\mathcal{S}$ and d is a pointer to a DOCDB object whose \$subject entry includes x.

Not shown in the figure are two archives: the $\mathcal{C}$- and the $\mathcal{D}$-archives. When we no longer anticipate any more queries on a deleted entity, we may, in the

[1] We assume that the schema would be maintained by a DBA, who would have to sanction all changes to it: a human operation meant to be slow and deliberate.

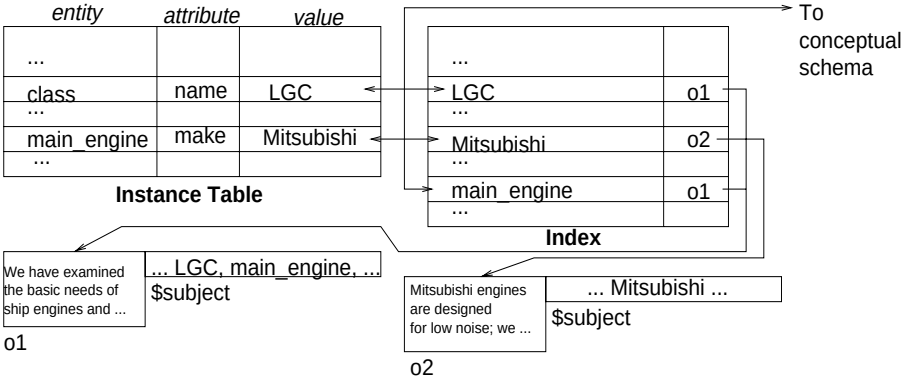

Fig. 2. The DOCDB. Objects o_1 and o_2 are document components. Object o_1's \$subject includes *LGC*, an instance of the attribute *name* of entity *class*, hence also an instance of that entity since *name* uniquely identifies each instance. It also includes a reference to the entity *main_engine* itself; this is recorded in the index but not in the instance table. Object o_2's \$subject includes *Mitsubishi*, an instance of the (inherited) attribute *make* of entity *main_engine*.

interest of future research, archive the final values of instances of that entity as it existed prior to deletion. These are listed in tabular form (a column for every useful attribute) in an an ASCII file and stored in the $\mathcal{C}$-archive. The instance table is periodically checked for instances that have been deleted; once a DOCDB object is found having \$subject entries consisting solely of deleted instances, it is automatically added to a $\mathcal{D}$-archive for manual inspection and possible re-classification. Also not shown is the schema $\mathcal{S}$ itself which is accessed by both writers and users.

5 The Writer's Role

We assume for simplicity that the writer is either reading an existing document or creating a new one focusing on one section at a time. We also assume that the writer is familiar with the language of the conceptual model. The system presents a graphical display of $\mathcal{S}$ and optionally the elements of $\mathcal{S}$ as a menu.

The writer begins by adding semantic information to the higher-level document component such as manual, chapter, or section, by appending to its \$subject list one or more references to elements in $\mathcal{S}$ or to actual instances of an entity or attribute of $\mathcal{S}$ (the writer selects the values from a displayed list). For example, suppose the writer indicates that a given subsection is about ships of class *Liquefied Gas Container (LGC)* and it also discusses main engines in general. The system would assign to the semantic attribute \$subject of that subsection object a list of two references: the first to an actual instance of the entity *class* with attribute *name* being *LGC*, and the second to an entity *main_engine*.

Through the hierarchy of document components, the lower ones inherit these attributes (they are filled in automatically by the system). For example, on filling in the value *hull* under $subject for a section of a manual, paragraphs objects inside this section object all inherit this attribute automatically. Queries can now locate such a paragraph as one about *hull* even though the word 'hull' never occurs inside it.

Of course, writers sometimes violate their own plan, including materials that are irrelevant, or, at best, distantly related to the main topic. A subcomponent could also be much more specialized than the topic defined in the section title, or it could focus on a completely different topic, for instance, to provide historical or conceptual background for what follows. In such situations, the writer will have to override the inherited attribute by adding to the $subject attribute, or modifying it. Because such pinpointing of a particular paragraph does not involve inserting huge lists of semantic attributes, the writer can focus on the few topics that differ from the general ones for the section or chapter.

Writers may also demand changes to the conceptual model. For example, the design history of a certain class of ships may be extremely interesting and there may be a great deal of information on that subject in documents; however, since it is not involved in the daily activities of the company, it neither occurs in CORPDB nor in $\mathcal{C}$. In this case, the writer asks for the addition of an attribute *design_history* [2] to the entity *ship*. This attribute becomes a part of $\mathcal{D}$.

Clearly the semantic information added depends on the diligence of the writers. But defining the topics to be covered in each component already makes up an important part of the work of the document design, before any writing or editing begins, and writers routinely devote a lot of thought to the question: what is this section going to be about? Writers will need to fill in a form defining $subject attributes for major sections, but after that, the writers may rely on the attributes trickling down in most cases; for instance, a chapter subject applies to all the large sections within that chapter, unless specifically turned off, added to, or modified. So the system does most of the work, and writers need only make a late-draft pass through identifying unusual or exceptional components. In addition, an editor could run a cursory check on the list of $subject entries for a portion of the document to ensure adequate breadth of coverage.

6 Coping with Changes

A very important real-world problem is that the enterprise changes a great deal. Changes occur as personnel come and go, processes become obsolete, and the corporation is reorganized. Such changes may make searches problematic. Our documents preserve semantic information about the enterprise, but the enterprise itself must discard concepts as they are outgrown, cancelled, or retired. Some documents, then, contain passages that describe concepts or properties

[2] This should not be confused with the history of the document, i.e., version control; that issue has been extensively studied by database researchers.

that no longer exist in the enterprise schema. Hence, a person who queries using only the current enterprise schema as criteria could return empty handed.

We cope with such inevitable change by allowing $\mathcal{C}$ and $\mathcal{D}$ to be modified according to an algorithm which, for shortage of space, we outline very briefly below. We use the shorthand *RID(x)* to mean *x is Referred In Document*, i.e., a writer has assigned x as $subject of a document component.

For $\mathcal{C}$, nothing needs to be done when it grows. However, when deleting a RID item, we must ensure that queries on such items are still viable: a deleted entity or attribute may still be queried for relevant documents. So we delete it from $\mathcal{C}$ and insert it into $\mathcal{D}$. Deletion of relationships have no effect.
Delete Attribute *a*: If RID(a) then move a from $\mathcal{C}$ to $\mathcal{D}$.
Delete Entity *e*: If e may be used in queries, move e from $\mathcal{C}$ to $\mathcal{D}$. Further, if e has been superseded[3], then create a mapping table for e
else if e should be archived, then create an e-table in the $\mathcal{C}$-archive.

For $\mathcal{D}$, the addition of new entities or attributes needs no work. If they are moving from $\mathcal{C}$ then their instance table entries already exist. Deletions from $\mathcal{D}$ of RID items are forbidden — writers are asked to reclassify all references to them before they can claim that the item is obsolete.
Insert Attribute *a*: Okay.
Insert Entity *e*: Okay.
Delete Attribute *a*: If a is being moved from $\mathcal{D}$ to $\mathcal{C}$, then okay.
Else if RID(a) then abort; else, clear its instance table (and index) entries.
Delete Entity *e*: *Similar to Delete attribute.*

7 Query Processing

Here we outline queries that can be processed using our approach. We state four queries in English and their translations in an MSQL-like syntax[4]. The first two queries rely on CORPDB while the last two rely on the schema $\mathcal{S}$. None of these can be processed by word search alone. The last one especially shows off the advantage of the use of a conceptual schema.

Q1 *Return section elements and included elements about ships built before 1980.* The information about delivery dates of ships are given precisely in CORPDB; hence a subquery off this database returns a list of ship names which can be used in DOCDB. Having obtained the relevant document elements, the OODB is exploited in finding enclosing section elements.

select section **from** DOCDB.D
where (D.component.$subject) **matches**
(**select** name **from** CORPDB.ship
where ship.delivery_date < 1980 **and** component ≤ section)

[3] An example is the case where departments are reorganized into divisions; queries on departments need to be re-mapped as queries on divisions.

[4] Users need not learn this syntax; they can use a more friendly *Query By Example* [19] interface.

Q2 *Return information about the design history of ships with Mitsubishi engines.*
This query also needs to get the information about which ships are fitted with Mitsubishi engines off CORPDB. This is then plugged in to DOCDB. Here, we assume that design history is in $\mathcal{D}$, and further that the query does not care which structural element gets returned so long as its $subject refers to design history and the relevant ships.

select * **from** DOCDB.D **where** (D.*.$subject) **matches**
'design_history' **and**
(**select** name **from** CORPDB.ship **where** ship **contains** c **and** c **in**
(**select** id **from** main_engine **where** make='Mitsubishi'))

Q3 *Return information about the design history of the ship 'Wanda'.*
This is a subquery of the previous one. It is based on the integrated schema which includes a *design_history* attribute for entity *ship*. This query does not need any information from CORPDB. It can be processed by looking up the index for references to design history and the ship named *Wanda*.

Q4 *Return information on painting of hulls of ships of class 'LGC'.*
For this query, we will outline how the problem indicated in the introduction can be addressed. The query does not need any information from CORPDB. However, looking up $\mathcal{S}$, it is clear that hull has a number of subcomponents. It is also apparent from the index of the OODB that at least one of the other parameters (painting) is relevant to one of the subcomponents (propeller). This leads to the query processor asking the user if information about painting of subcomponents of hulls is useful (in the case of generalization/specialization hierarchies, this user interaction can be skipped). On an affirmative answer, the crucial information about the propeller would be retrieved. This shows the power of the conceptual model per se.

8 Related Work

Modeling of document databases has long been studied by researchers [6], [9], [10], who have mainly explored the hierarchical nature of the structure of documents. Extending the relational database model by augmenting SQL with data types was suggested by [3]. The appropriateness of the object-oriented model for SGML has been demonstrated; [5] mapped an SGML DTD into class definitions. Our work can be viewed as an extension of theirs: we add semantics.

It has been recognized that SGML is flexible — enhancing the functionality of SGML can enhance the functionality of the final product [18]. Hence there have been efforts to add enhancements to SGML documents augmenting the scope of queries; three approaches are important. In the first [7], an ER schema is presumed to exist and relevant instances of entities and relationships are included in documents using special tags. The resulting documents are said to form a *lightweight database* because database-like searches are possible on them. We prefer a stronger insulation between the fast-changing enterprise database and the more gradually evolving collection of documents. In the second approach

[18], reference links are added from SGML documents to an existing database. When a document is looked up, certain character strings get translated into values from the database. They neither deal with semantics, nor with changes in the enterprise. Our work is orthogonal to theirs: we do not alter the document content. The last one [15] builds an outlining tool for technical documentation by adding semantic text as we do. However, they avoid semantic models and are consequently very restricted in their domain of applicability.

The need for semantics of documents has also been recognized by researchers in information retrieval [17]. The MIRACLE system [13], based on the INQUERY engine [4] exploits conceptual models. However, its primary means of document retrieval remains probabilistic based on occurrences of certain words and phrases. Moreover, use of the conceptual model is primarily geared towards expansion of user queries. While this allows more freedom to the user, the queries still need disambiguation because it is simply impossible to connect all possible user terms with all corresponding phrases occurring in documents. In our approach, the conceptual model is the fundamental infrastructure for both writers and users and is visible to both. In a sense, we place more faith in the user to exploit a conceptual model and pose queries and modify them according to their responses.

The GRAM system [1] attaches semantics to linked hypertext documents by placing each document in a class specified by a few attributes. These classes are nodes in a graph with edges providing the meaning of links between such documents. A path represents navigation traversing links; a path algebra provides a query language to users. While this approach provides clarity to hypertext browsing, it is not adequate for huge detailed collections of documents such as the suite of manuals for a ship. GRAM can be a good front-end for a hypertext version of the documents in our DOCDB.

9 Conclusion

There are several advantages of our approach:

- Our approach scores over automatic indexing and probabilistic search [20] (note that we do not outlaw such searches, but augment these methods) because we empower a human writer to clearly define the semantics of document pieces instead of relying only on probabilistic word or phrase matching.
- Our approach is viable and useful for writers. First, it does not require them to make ad-hoc manual changes to a DTD (in our approach, the DTD is changed uniformly and automatically). Second, they are not asked to associate semantics with *every* low level structural element of documents such as paragraphs — a massive and unrealistic task; instead, we exploit the natural inheritance accompanying the document's structural hierarchy. Third, they can make sequential read-throughs of documents; reused document portions reuse the semantic attributes labeled earlier. Fourth, the semantic attributes are helpful in validating links to portions of other documents.

- The conceptual schema is a bridge between the writer and the user; it guides the writer away from arbitrary phrases or codes towards terms whose relevance and meaning the user is fully aware of. The user's awareness of the concepts and conceptualization is key to the viability of any information system.
- Enhanced query support arises from the semantic information added, the power of the conceptual model used, and use of the information in the enterprise database. The absence of any of these elements merely detracts from the full power but does not invalidate the method. In the worst case, when none of these are available, our approach degenerates into the word based search systems.
- In most enterprises, the world of documents remains divorced from routine database operations. As a result, users have to anticipate the source of information of a given query; for example, while a university's student database typically contains the schedule of courses to be offered in the next semester, the syllabi of those courses are kept elsewhere. A single integrated schema for the entire enterprise would be very desirable because of the powerful and comprehensive queries that it would enable. This is exactly what our approach achieves while avoiding the pitfalls of post-hoc schema integration where one creates a separate schema from a new document database and attempts merger with that of the enterprise.

Our approach faces a problem in very large corporations with heterogeneous databases where it is very hard to arrive at a *single* conceptual schema $\mathcal{C}$. Of course, this is a hurdle faced by any information system trying to span the whole organization. Typically, the separate autonomous databases belong to different groups and each group has its own world of documents; thus our approach works with the individual groups. When company-wide documents need to be created by Headquarters, each group can export a small part of their schema; our approach then works for Headquarters' document database using the collection of these exported schemas.

To summarize, we have outlined a methodology for writers to add semantic information to a document database which gets integrated with the traditional conceptual schema of the enterprise, thus creating an information space of the entire organization, over which powerful queries are enabled. We have implemented parts of this approach using the ObjectStore OODBMS; a complete working tool is ongoing work.

Acknowledgments We are indebted to anonymous referees for comments and questions that have helped to improve the paper. Partial support has been provided to the first two authors by the National Science Foundation under contract IRI-9509789 and to the first and third by Sandia National Laboratories.

References

1. B. Amann and M. Scholl. GRAM: A Graph Data Model and Query Language.

In *Proceedings of the Fourth European Conference on Hypertext and Hypermedia ECHT'92.*, pages 201–211, 1992.

2. C. Batini, S. Ceri, and S. Navathe. *Conceptual Database Design.* Benjamin/Cummings, 1992.
3. G. Blake, M. Consens, P. Kilpeläeinen, P.-Å. Larson, T. Snider, and F. Tompa. Text/Relational Database Management Systems: Harmonizing SQL and SGML. In W. Litwin and T. Risch, editors, *Applications of Databases, Proceedings of the First International Conference ADB-94*, pages 267–280. Springer-Verlag, 1994.
4. J. Callan, B. Croft, and S. Harding. The INQUERY Retrieval System. In *Proceedings of the Third International Conference on Database and Expert Systems Application*, pages 78–83, 1992.
5. 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, Minneapolis, Minnesota*, pages 313–324, 1994.
6. W. B. Croft and D. W. Stemple. Supporting Office Document Architectures with Constrained Types. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data, San Francisco, California*, pages 504–509, 1987.
7. S. Dobson and V. Burrill. Lightweight Databases. In *Proceedings of the Third International World Wide Web Conference*, 1995.
8. C. Goldfarb. *The SGML Handbook.* Clarendon Press, Oxford, 1990.
9. R. H. Guting, R. Zicari, and D. Choy. An Algebra for Structured Office Documents. *ACM Transactions on Office Information Systems*, 7(4):123–157, April 1989.
10. M. Gyssens, J. Paredaens, and D. Van Gucht. A Grammar-based Approach towards Unifying Hierarchical Data Models. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Portland, Oregon*, pages 263–272, 1989.
11. ISO8879:1986. Information Processing — Text and Office System – Standard Generalized Markup Language (SGML), 1986.
12. W. Kim, H. Chou, and J. Banerjee. Operations and Implementation of Complex Objects. In *Proceedings of the IEEE Third International Conference on Data Engineering*, 1987.
13. A. Müller and U. Thiel. Query Expansion in an Abductive Information Retrieval System. In *Proceedings of the RIAO'94. New York.*, pages 461–480, 1994.
14. J. Price. Introduction: Special Issue on Structuring Complex Information for Electronic Publication. *IEEE Transactions on Professional Communication*, 40(2):1–9, June 1997.
15. C. Tattersall and A. Cole. Modelling the Content of Technical Documentation. In *Proceedings of Electronic Publishing EP-92*, pages 223–232, 1992.
16. E. van Herwijnen. *Practical SGML.* Kluwer Academic, 1994.
17. C. van Rijsbergen. Towards an Information Logic. In *Proceedings of the Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1989.
18. M. Yoshikawa, O. Ichikawa, and S. Uemura. Amalgamating SGML Documents and Databases. In *Proceedings of the 5th International Conference on Extending Database Technology*, pages 259–274, 1996.
19. M. Zloof. Query By Example: a Data Base Language. *IBM Systems Journal*, 16(4):324–343, 1977.
20. J. Zobel, A. Moffat, and R. Sacks-Davis. An Efficient Indexing Technique for Full Text Databases. In *Proceedings of the Eighteenth International Conference on Very Large Databases*, pages 352–362, 1992.