Exhibit 3
To Third Declaration of
Joseph N. Hosteny

The ELF data model and SGQL query language for structured document databases

T. Arnold-Moore *M. Fuller* *B. Lowe* *J. Thom* *R. Wilkinson*
Department of Computer Science
RMIT
Melbourne
Australia

tja@citri.edu.au *msf@citri.edu.au* *lowe@citri.edu.au* *jat@citri.edu.au* *ross@citri.edu.au*

Abstract

*A data model and query language for accessing structured documents expressed in SGML is presented. The ELF (***EL***ements with* **F***eatures) model uses the SGML grammar (DTD) directly as a schema avoiding transformations which can lose information. The model also gives flexibility to the implementor to retrieve whole documents and decompose them, retrieve atomic elements and recombine them, or pursue alternatives which retrieve the elements directly. The language,* **S***tructured* **G***eneralized* **Q***uery* **L***anguage (SGQL), allows efficient access to the content, structure and attributes of documents at any level within their structure. This is all achieved with a simple, largely orthogonal functional language.*

1 Introduction and motivation

Document query languages have most commonly been developed for information retrieval systems. In these systems, documents have been most commonly very small. Queries might specify that certain attributes have particular values, but have concentrated on matching against the content of the document. Thus we might have the query expressed in Common Command Language (CCL) [11]:

```
FIND au = Smith AND
ab = (document AND database)
```

Query 1

which finds records containing 'Smith' in the author field and both 'document' and 'database' in the abstract field.

Rather than exact matching, it has been found to be more effective [27] to use a ranking formula that measures the *similarity* of the query to the document, so that the query might simply be:

```
"document database"
```

Query 2

which is then compared to the content of each document, and the documents are then returned in order of their similarity[20]. These methods have proved to be very effective for the retrieval of large collections of abstracts or short articles.

Another type of system that has achieved significant use is the hypertext system. In this case, information is typically broken into small units. This information is explored by browsing, rather than querying, however we may view the traversal of a link as another kind of query. In fact some authors have proposed that links are in fact instantiated as queries.

A new class of document databases is emerging. These databases consist of large structured documents. Examples include databases of government legislation, maintenance manuals for systems as complex as aircraft carriers, an encyclopedia, and the documentation associated with a large software engineering project. These databases are inadequately catered for by current database systems.

These databases will need to be searched by attribute. This will, for example, allow a software engineering document that is the right version to be retrieved. The databases will need to be queried on content. However the nature of very large documents may make a query on whole documents inappropriate. It may be appropriate to search for "metal fatigue" only in the sections that are about "wings". The databases should also allow for partial document retrieval. The whole of a government Act may be an inappropriate retrieval unit, if one is searching for a definition. There may be a number of relevant portions of a single document that are relevant, and yet the whole document may still be an inappropriate retrieval unit. However in a different context, the whole document may be exactly the right retrieval unit. Finally, we will certainly wish to follow any hypertext links that are provided. This may mean that appropriate table of contents support is required also.

We thus see that a database system to support databases of large structured documents need a query language that allows retrieval:

- by exact matching Boolean combinations of words and phrases;
- by ranking by similarity to a given text;
- using hypertext links;
- by attribute;
- of and by arbitrary sub-parts or whole documents.

Use of grammars to describe the structure of documents has long been associated with databases and many previous dealings with structured documents have constructed their own grammar [8, 24, 25]. The ISO standard, Standard Generalized Markup Language (SGML) [10], now provides a grammar for describing document structure which is widely used for document exchange.

SGML describes a tagging scheme and a metagrammar for describing the structure of documents. Each document *instance* consists of a *declaration* (which describes the character set and the available facilities), a *DTD* (document type definition – the grammar which the document satisfies) and the tagged text itself. Standard declarations and DTD's can avoid the need to transfer this information unnecessarily but the power exists to describe unusual structures or documents.

Each structural *element* in the body of the text is surrounded by a begin (`<gi>`) and end (`</gi>`) tag where `gi` is any *generic identifier* or element type name. Elements can be nested within each other and the DTD allows the specification of the *content model* of each element type, with quite complex combinations easily expressed.

One can associate typed information with particular SGML elements by using *attributes* which appear in the text within the begin tag. Because of its wide-spread usage and expressive power, considerable work has been put into translating plain text and structured text into SGML [4, 6, 29] so even those documents not currently in SGML can be converted. A complete description of SGML can be found in [7].

Given that SGML provides a way of describing all of the meaningful fragments of a document, we may modify and clarify the desired list of features in order to more precisely describe the characteristics of a structured document query language:

- query across different documents;
- return lists of SGML elements;
- query on content;
- query on SGML and non-SGML attributes;
- query on pure structure; and
- query on a mixture of content and SGML structure.

We nearly have what we desire. However, we may not be able to rely on the designer of the SGML document class to take into account all of the attributes that may be appropriate to query. Thus, we may have associated with each document, or element of a document, a set of attributes that we shall call features, to distinguish them from the attributes defined by the grammar that describes the SGML document class.

In this paper we discuss some representative queries which illustrate the requirements of a query language for handling structured documents. After discussing related work on models for structured document database, we propose a data model specifically designed for storing large numbers of complex structured documents, and discuss a language in a functional notation showing how the representative queries can be expressed. Finally we discuss the strengths and weaknesses of the model and areas for further work.

2 Representative queries

In order to focus our exploration of the needs of a structured document query language, we discuss some representative queries which illustrate these key requirements.

2.1 On content

The most widely used repositories of text are all based on boolean retrieval of complete documents. The use of the vector-space model for ranked queries is also achieving wide acceptance [23] and is demonstrating better retrieval performance than the traditional boolean model [27]. Therefore both must be supported leading to the following queries:

Find docs about 'keyword' AND 'keyword2'.

Query 1.1

Find docs similar to 'keyword' *with measure above 0.5.*

Query 1.2

2.2 On content and structure

Salton's more recent work has dealt with smaller fragments than documents [21, 22]. His success suggests the following queries (SGML versions of those suggested by Salton):

Find <section>'s similar to 'keyword'.

Query 2.1

Find <section>'s with (<par>'s similar to 'keyword' *with measure above 0.2).*

Query 2.2

Find docs similar to 'keyword' *with (<section>'s similar to* 'keyword' *with measure above 0.5).*

Query 2.3

Unlike Salton's work where the level was fixed at database creation time, SGML's power could be utilized to extract elements at any arbitrary level providing efficient indexing schemes can be devised [12, 18, 30].

2.3 Across different documents

Ideally we wish to query across documents satisfying a number of DTD's which may use different generic identifiers to describe logically similar units. The use of a macro facility can overcome this [16]:

Find articles and memos with TITLE's about 'keyword'.

Query 3.1

where TITLE is a macro defined for each DTD describing what element(s) should be viewed by the user as being titles.

2.4 On attributes

Object-based query languages get much of their expressive power from the ability to test the contents of attributes associated with the objects. Hypertext links are often implemented in object systems and SGML using attributes. SGML provides a facility to associate attributes with elements (the analogue of objects) within the text itself and some facility to query on these is needed:

Find docs with attribute CONF.

Query 4.1

Find <author>'s of docs with attribute CONF = 'yes'.

Query 4.2

It may be necessary for the database to store additional information which does not form part of the text of the SGML document instance. These might include querying on the database document identifier:

Find doc with identifier '92'.

Query 4.3

or querying on the document class (the DTD which describes the content model of the element):

Find docs satisfying the 'memo' *DTD.*

Query 4.4

2.5 On structure

Already we have described queries which mix structure and content. Queries which allow examination of the structure of the document are also necessary to take full advantage of the additional information available in the SGML tagging:

Find docs with 8 <section>'s.

Query 5.1

Find siblings/children/parent of this element.

Query 5.2

3 Related work

The need for more sophisticated database systems for handling structured documents has been recognized by numerous researchers. A number of data models have been put forward to solve these problems. These issues and the related issue of indexing schemes to support such queries are discussed in greater detail elsewhere [19] however a brief summary is appropriate.

Rossiter and Heather [18] provided an early survey of the potential of various models to support access to structured documents. They considered the free-text, relational, semantic and object-based models.

The free-text model, that used in most traditional text-retrieval systems, treats documents as lists of words disregarding any internal structure in the documents thus supporting only boolean and ranked queries on the content of whole documents.

The relational model extended to support content queries can support a whole range of queries including mixed content and structure, pure structure, and attribute queries. However representation of complex documents by mapping complex elements onto tables leads to large numbers of tables and tuples [14]. Even simple queries, particularly queries on the content of whole documents, require many join operations as global indexes are hard to build [18]. These join operations can be costly to compute, particularly in larger databases. Queries across databases containing documents with different structures are not well supported by this model as different document types require separate tables. The semantic models discussed by Rossiter and Heather are simply variations on the relational model and suffer similar drawbacks.

Their preferred option was to use an object-based model, mapping every element onto an object in the database. This model also supports a wide range of queries [2, 5, 14] but again, simple queries require many join operations. Each document type requires a separate class definition creating problems for queries involving more than one document type.

Some approaches not considered by Rossiter and Heather include the field model, and element-based models. A refinement of the full-text approach is to define fields [11] or zones [9, 14] allowing queries to be limited to the content of specified zones. These zones are typically defined by the database administrator when the database is constructed. SGML can be supported by using element tags to specify the boundaries of these zones. Queries involving multiple document types can be supported by mapping the appropriate elements in each document type to a single field, e.g. the 'au' field (short for author) might be defined to correspond to the `<author>` element in a DTD for journal articles and the `<from>` element in a DTD for electronic mail or office memos. The disadvantage of this approach is that either the fields for which query support is provided must be decided at database creation time, or that a field be defined for every element in the DTD. As this model is typically supported by constructing an index for each field, this usually results in huge storage requirements just for the indexes with substantial overlap in coverage of the indexes. This model supports boolean or ranked queries based on content for whole documents and for parts of documents and can support the retrieval of whole or part documents.

Each of these models involves mapping the DTD description of the document structure onto a schema for the appropriate model. Such a translation process is likely to lose information, potentially ignoring some constraints imposed by the DTD that are unable to be represented by the model's schema definition mechanism. These models also rely on retrieving minimal parts of documents and recombining them (relational, semantic and object models) or by retrieving whole documents and decomposing them (full-text and field models). What is needed is to support retrieval by element directly using the DTD as the schema.

One example of this approach allows the definition of columns in a relational table containing structured text [3]. The structure of the text is defined only by reference to the element name and the DTD which defines its structure. Access is supported by adding an EXPAND operator to SQL which applies only to columns of structured text. Within the scope of the EXPAND operator, three virtual tables are defined:

```
TEXT_NODES(nodeid, genid, content)
TEXT_SRUCTURE(a_nodeid, d_nodeid)
TEXT_ATTRIBUTES(nodeid, attr, value)
```

In the first table we find a single tuple for each element with a unique identifier (`nodeid`), the type of the node (generic identifier–`genid`) and the text which makes up that node. The second table represents child-parent relationships between nodes and the last gives access to the SGML attributes. The writers are careful to stress that these tables are *virtual* and need not be stored or indeed ever created. If they are not actually present then the model suggests that documents will be decomposed at query time to extract the relevant information. If they are present, direct support of element retrieval is provided but costly join operations still need to be performed for structure queries which rely on the `TEXT_SRUCTURE` table.

4 Data model

We propose an alternative data model for structured text which relies solely on the DTD to provide a schema for the data and supports element access directly without join operations.

In order to construct a conceptual model of the database system we consider the database to be a list of **ELF**'s (ELements with Features) where an **ELF** is:

- a complete SGML element – primitive content tokens (#PCDATA, CDATA and RCDATA) are considered to be elements for this purpose;
- a list of features associated with that element (we avoid using the term 'attribute' to prevent confusion with SGML attributes).

We consider every element in every document to be in this list although it may be necessary to provide the database administrator with a mechanism for specifying a granularity beyond which we do not represent sub-elements. In general, the result of a query will be a list containing a subset of the ELF's in the database. While we allow some queries to return a numeric result or a boolean value for use in other queries, orthogonality is maintained since all queries can be used as sub-queries. The element features required include:

EID (an absolute element identifier). When versioning hypertext, links can either be static or dynamic [15]. In order to support static links to elements we require an absolute identifier for each ELF. The EID is also useful for supporting dynamic inclusion of sub-elements [13]. The EID should be allocated by the DBMS and should be unique for each element even though the element may occur in more than one version tree and once allocated should not change.

DID (a document identifier). Every version of an SGML document in the database should also have a unique, absolute identifier which is assigned by the DBMS. For elements which are SGML documents, EID = DID. Where elements may appear in more than one document, a list of DID's will be required.

DTD (the DTD which that document satisfies). Every document from which an element comes will satisfy a DTD and this information should not be stored more than once in a particular database. Instead of storing the DTD with each occurrence of a document of that type, maintain an identifier (or simply a pointer) to the relevant DTD. This information will be needed at the application level for browsers and editors to access.

OS (an associated output specification e.g. a FOSI [28] or style file [31]).

SIM (a similarity measure) In order to support ranked queries, we need some way of associating a similarity measure with each ELF in the list being ranked. A separate feature satisfies this requirement. For queries that are not ranked, this value can be set to some default (0 or 1 or negative) to demonstrate its invalidity.

LOC (the location of that element within the document). Some method of identifying where the element is located within a particular document is needed. Alternatives include those provided in HyQ [12, 17] and the REL indexing scheme [1]. If an element may appear in more than one document, a list of LOC's will be required corresponding to the list of DID's.

Generally functions operate on lists of elements as a list is the simplest structure which will return a possibly ranked ordered collection of documents. We choose elements as our base rather than whole documents as an SGML document is always an element, and using elements adds generality to the query without undue additional complexity allowing arbitrary node sizes instead of the traditional fixed node size.

In order to avoid the *ad hoc* creation of DTD's at query time for exporting or displaying partial sub-trees, only complete elements can be exported. Their structure is defined already by the DTD for the document to which they belong. Mixed content models can be dealt with by assuming that the primitive data tokens can themselves be elements [7]. A table of contents for a given document will constitute a partial sub-tree if treated as a single structure. However each of the entries in the table of contents will be a valid element (or list of elements) in an existing DTD. Since an ordered list is returned, when a table of contents is requested for a particular document, the database can simply return a list containing all the elements corresponding to each entry in the table in the order that they appear in the document. The DID feature can be used to produce tables for more than one document.

Each element within a document will have many of these features in common with other elements in the same document. The physical model to support the interface provided by this logical model should reflect this commonality to prevent unnecessary duplication.

5 The SGQL Language

We describe a new functional query language, SGQL (Structured Generalized Query Language) over this data model by describing the data types and a set of functions on particular data types. Figure 1 shows the valid types for arguments and return values, Figure 2 shows the functions which return lists of **ELF's** and Figure 3 shows other functions in SGQL. We demonstrate the utility of SGQL by expressing the representative queries from Section 2 in SGQL.

5.1 On content

Boolean queries are supported by the `contains` function whereas ranking queries are supported by the functions — **rank** (rank all elements), `rank_t` (rank only those elements above a threshold similarity) and `rank_f` (rank only the N most similar documents). The following is an example of a boolean query:

Find docs about 'keyword' *AND* 'keyword2'.

Query 1.1

In SGQL this query would be expressed as follows:

```
contains(has_gi(*, DOC),
          "keyword1" & "keyword2")
```

where `has_gi(*, DOC)` is the list of all documents, that is elements which have the generic identifier specified by the macro `DOC` (`DOC` behaves like a list of <gi>'s and is defined for each DTD describing what element should be viewed by the user as being a whole document).

Ranking queries are handled in a similar way:

Find docs similar to 'keyword' *with measure above 0.5.*

Query 1.2

This query uses `rank_t` with the additional argument specifying the threshold of 0.5.

```
rank_t(has_gi(*, DOC), "keyword", 0.5)
```

*	All elements present in the database.
ELF	An SGML Element with attached Features.
[ELF]	A (possibly empty) list of **ELF**.
<gi>	An SGML Generic Identifier.
[<gi>]	An (possibly empty) list of <gi> (comma separated).
feature name	The name of a Feature of an **ELF**.
attribute name	The name of an Attribute of an **ELF**.
F	The value of a Feature of an **ELF**.
A	The value of a Attribute of an **ELF**.
PEXPR	A phrase expression.[1]
BOOL	A Boolean combination of **true**, **false**, and Boolean functions (see below).
string	0 or more ascii characters, delimited by ".
N	An Integer or Real.
R	A relationship specifier.

Figure 1: Data types in SGQL.

contains	**[ELF]** × **PEXPR**→ **[ELF]**
rank	**[ELF]** × **PEXPR**→ **[ELF]**
rank_f	**[ELF]** × **PEXPR**× N → **[ELF]**
rank_t	**[ELF]** × **PEXPR**× N → **[ELF]**
test	**[ELF]** × **BOOL**→ **[ELF]**
has_gi	**[ELF]** × [<gi>] → **[ELF]**
intersect	**[ELF]** × **[ELF]** → **[ELF]**
union	**[ELF]** × **[ELF]** → **[ELF]**
subtract	**[ELF]** × **[ELF]** → **[ELF]**
unique	**[ELF]** → **[ELF]**
top	**[ELF]** × N → **[ELF]**
maximal_elt	**[ELF]** → **[ELF]**
relation	**[ELF]** × **[ELF]** × R → **[ELF]**

Figure 2: SGQL functions returning **ELF**'s.

count	**[ELF]** → N
get_attribute	**ELF**× attributename → A
get_feature	**ELF**× featurename → F
equals	expression × expression → **BOOL**
greater	expression × expression → **BOOL**
lesser	expression × expression → **BOOL**
and	**BOOL**× **BOOL**→ **BOOL**
or	**BOOL**× **BOOL**→ **BOOL**
not	**BOOL**→ **BOOL**

Figure 3: Other SGQL functions.

5.2 On content and structure

Queries can also be made on elements other than whole documents:

Find <section>'s similar to 'keyword'.

Query 2.1

In SGQL we can specify that we want to query `section` elements rather than whole documents by specifying those elements whose generic identifier is <section>, that is `has_gi(*, [<section>])`.

```
rank(has_gi(*, [<section>]), "keyword")
```

The use of the `rank` function means all sections will be returned ranked on their similarity to the phrase "keyword".

The `relation` function allows queries at different levels within a document, as in the following example:

Find <sections>'s with (<par>'s similar to 'keyword' *with measure above 0.2).*

Query 2.2

In SGQL the sub-query:

```
has_gi(*, [<section>])
```

returns all sections, and the sub-query:

```
rank_t(has_gi(*, [<par>]),
       "keyword", 0.2)
```

returns all paragraphs with a similarity to *'keyword'* above 0.2. These results can be be combined by using the `relation` function to return only those sections that contain a paragraph similar to *'keyword'* — that is in the parse tree of the document the sections must be an `ancestor` of a matching paragraph.

```
relation(has_gi(*, [<section>]),
         rank_t(has_gi(*, [<par>]),
                "keyword", 0.2),
         ancestor)
```

Other relationships in the parse tree can also be tested, including `parent`, `child`, `descendent`, and `sibling`. The following query is another example of the use of the function `relation`:

Find docs similar to 'keyword' *with (<section>'s similar to* 'keyword' *with measure above 0.5).*

Query 2.3

In SGQL the query is similar to the previous one except that this time it is embedded in an outer rank function.

```
rank(relation(has_gi(*, DOC),
              rank_t(has_gi(*,
                            [<section>])
                     "keyword",
                     0.5),
              ancestor),
     "keyword")
```

5.3 Across different documents

The macro facility can also be used to query across databases containing many different types of document:

Find articles and memos with TITLE's about 'keyword'.

Query 3.1

where TITLE is a macro defined for each DTD describing what element(s) should be viewed by the user as being titles; it behaves like a list of <gi>'s. This query also demonstrates that the `relation` function can also be used with boolean sub-queries. If we assume the database contains only articles and memos, in SGQL the query is as follows.

```
relation(has_gi(*, DOC),
         contains(has_gi(*, TITLE),
                  "keyword"),
         ancestor)
```

5.4 On attributes

Queries can be made on both the SGML attributes associated with elements (using the function — `get_attribute`) and on the additional SGQL features that are also associated with elements (using `get_feature`). The following is an example of query on an SGML attribute:

Find docs with attribute CONF.

Query 4.1

```
test(has_gi(*, DOC),
     not(get_attribute(test_elf, "CONF")
         = NULL))
```

In SGQL the function `test` returns a (possibly empty) list of elements for which the specified is true. The variable `test_elf` represents the current element to be tested and is available within the second argument of `test`.

While the above query tested for the existence of an attribute, the following query tests the value of an attribute:

Find <author>'s of docs with attribute CONF = 'yes'.

Query 4.2

In SGQL we need to use the `relation` function to find the authors within the matching documents.

```
relation(has_gi(*, [<author>]),
         test(has_gi(*, DOC),
              get_attribute(test_elf,
                            "CONF")
              = "yes"),
         child)
```

Queries on features are implemented in a similar way. For example the following query on the database document identifier (DID):

Find doc with identifier '92'.

Query 4.3

In SGQL this can either be expressed as

```
test(has_gi(*, DOC),
     get_feature(test_elf, DID) = "92")
```

or alternatively as

```
maximal_elt(get_feature(*, DID) = "92")
```

The function `maximal_elt` only returns those elements which are not sub-elements of other elements in the list.

Another feature of documents and elements is the document class (that is the DTD which describes the content model of the element) suggesting the following query:

Find docs satisfying the 'memo' *DTD.*

Query 4.4

```
test(has_gi(*, DOC),
     get_feature(test_elf, DTD)
     = "memo")
```

5.5 On structure

We have already introduced the `relation` function which enables complex queries to be expressed on the structure of documents. For example, the following query needs to count the number of elements of a particular type in a document:

Find docs with 8 <section>'s.

Query 5.1

In SGQL this would be expressed as follows.

```
test(has_gi(*, DOC),
     count(relation(has_gi(*,
                           [<section>]),
                    test_elf,
                    descendant))
     = 8)
```

The following is a more complex example:

Find siblings and children of this element.

Query 5.2

where *this element* is specified by DID '92' and LOC '1-3-2-5-2'. The `union` function allows two answer lists to be combined in a similar way to the relational union operator.

```
union(
 relation(*,
          test(*,
               get_feature(test_elf, DID)
               = "92" and
               get_feature(test_elf, LOC)
               = "1-3-2-5-2"),
          sibling),
 relation(*,
  test(*,
               get_feature(test_elf, DID)
               = "92" and
               get_feature(test_elf, LOC)
               = "1-3-2-5-2"),
          child))
```

6 Conclusions

We have presented a data model for structured documents which does not require any additional schema to be defined other than the DTD which must already be defined for any SGML document. The model does not limit an implementor to either decomposing whole documents, or reconstructing larger portions of text from atomic elements at query time. By treating the database simply as a list of elements (with associated features) the implementor can use either of these approaches. The inclusion of the location feature also allows the implementor the freedom to explore alternative approaches to retrieval of elements.

We have demonstrated that a wide variety of interesting queries can be easily expressed in SGQL. This arises from the deliberate effort to describe a model specifically for structured documents. Because of the inherent orthogonality of the language and strict limits on the return types of functions, SGQL can be used to provide a large number of facilities with a small number of functions.

While the functional notation may not be as intuitive to the casual user as something like SQL, it is anticipated that SGQL will be primarily used as an API to text and graphical user interfaces rather than used directly by the user. It is presumed that these interfaces will have access to the appropriate DTD's and output specifications so that users will be able to avoid knowing the exact generic identifiers required for every query. The functional notation also has an additional advantage over more declarative languages in that implementation issues are rather more immediate.

SGQL provides querying capabilities on the content of documents, or the attributes of documents, or on the structure of documents, or on any combination of these. The use of SGML as the underlying grammar gives the user access to the underlying structure without sacrificing authors' flexibility to create new document types with relative ease. It also has the advantage of access to a large body of existing text in SGML and sophisticated tools for the conversion to and from other formats.

The functions described in this paper are purely for retrieval purposes. Further work needs to be done on SGQL to describe appropriate data manipulation functions.

Additional improvements, principally in the abbreviation of some queries (and more efficient implementation without optimization) could be achieved by providing a variable facility to avoid duplication within queries. In particular query 5.5 could be substantially improved.

The problem of combining similarity measures has received little attention. Query 5.2 uses the similarity of <section>'s as a filter of documents but the ranking of the documents is done by the similarity measure of the whole document. The ability to express queries such as:

Rank (docs similar to 'keyword' *with (<par>s similar to* 'keyword2'*)).*

Query 3

is not yet provided. Whether the user or even the database administrator should have the power to control how this is done is a question which requires considerable investigation.

This query language focuses on the needs of structured text documents without significant consideration of multimedia issues of inclusion of sound and pictures (be they still or moving). There is no reason to limit SGML to handling just text (as HyTime illustrates [17]) however SGML provides the facility to refer to external objects which are non-SGML. These could be included in the database by wrapping them in a database-defined markup so that they satisfy the requirements of an ELF.

References

[1] Timothy Arnold-Moore and Ron Sacks-Davis. The Relative Element Locator scheme for indexing SGML documents. Technical Report CITRI TR/94-3, Collaborative Information Technology Research Institute, 1994.

[2] E. Bertino, F. Rabatti and S. Gibbs. Query processing in a multimedia document system. *ACM Transactions on Office Information Systems*, Volume 6, page 1, 1988.

[3] G.E. Blake, M. P. Consens, P. Kilpeläinen, P.A. Larson, T. Snider and F.W. Tompa. Text/relational database management systems: Harmonising SQL and SGML. In *Proc. Int. Conf. on Applications of Databases*, page 267, Vadstena, Sweden, June 1994.

[4] Mark H. Chignell, Bernd Nordhausen, J. Felix Valdez and John A. Waterworth. The HEFTI model of text to hypertext conversion. *Hypermedia*, Volume 3, page 187, 1991.

[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 the Management of Data*, page 313, Minneapolis, MN, 1994.

[6] Exoterica Corporation. *XGML Omnimark Version 2.2*, 1993. Solaris 2 or MS-DOS.

[7] Charles F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.

[8] G. Gonnet and F. Tompa. Mind your grammar: a new approach to modelling text. In *Proceedings of the Very Large Data Bases Conference*, page 339, Brighton, 1987.

[9] G. H. Gonnet et al. Lexicological indices for text: inverted files vs. PAT trees. Technical Report OED-91-01, University of Waterloo, Ontario, Canada, 1991.

[10] International Organization for Standardization. Information processing - text and office systems - standard generalised markup language (SGML), 1986. ISO/IEC 8879:1986.

[11] International Organization for Standardization. Information and documentation - Commands for interactive text searching (CCL), 1993. ISO/IEC DIS 8777:1993.

[12] W. Elliot Kimber. HyTime and SGML: understanding the HyTime HyQ query language. Technical Report E14/B500, IBM Corporation, 1993.

[13] Sauro Lamberti, Cesare Maioli and Fabio Vitali. Some modifications to the Dexter model for the formal description of hypertexts. Technical Report UBLCS-93-5, Laboratory of Computer Science, University of Bologna, Laboratory of Computer Science, University of Bologna, Piazza di Porta S. Donato, 5 40127 Bologna Italy, April 1993.

[14] I. A. Macleod. Storage and retrieval of structured documents. *Information Processing and Management*, Volume 26, Number 2, page 197, 1990.

[15] Cesare Maioli, Stefano Sola and Fabio Vitali. Versioning issues in a collaborative distributed hypertext system. Technical Report UBLCS-93-6, Laboratory of Computer Science, University of Bologna, Laboratory of Computer Science, University of Bologna, Piazza di Porta S. Donato, 5 40127 Bologna Italy, April 1993. Available by ftp from `ftp.cs.unibo.it` in pub/TR/UBLCS.

[16] Multimedia Database Systems Group. *SIM Data Definition Language and Database Administrators Guide.* Melbourne, 1994.

[17] S. R. Newcomb, N. A. Kipp and V. T. Newcomb. 'HyTime' the hypermedia/time-based document structuring language. *Communications of the ACM,* Volume 34, page 67, 1991.

[18] B. N. Rossiter and M. A. Heather. Strengths and weaknesses of database models for textual documents. In *Proceedings of Electronic Publishing '90,* page 125, Gaithersburg, MD, 1990. Cambridge Uni Press.

[19] Ron Sacks-Davis, Timothy Arnold-Moore and Justin Zobel. Database systems for structured documents. In *Proceedings of the International Symposium on Advanced Database Technologies and Their Integration (ADTI),* page 272, Nara, Japan, October 1994.

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

[21] Gerard Salton, James Allan and Chris Buckley. Approaches to passage retrieval in full text information systems. In *Proceedings of the ACM/SIGIR International Conference on Research and Development in Information Retrieval,* page 49, Pittsburgh, PA, 1993.

[22] Gerard Salton, James Allan and Chris Buckley. Automatic structuring and retrieval of large text files. *Communications of the ACM,* Volume 37, Number 2, page 97, 1994.

[23] Gerard Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Tokyo, 1983.

[24] F. Sarre and U. Günzter. Automatic transformation of linear text into hypertext. In *International Symposium on Database Systems for Advanced Applications,* page 498, Tokyo, 1991.

[25] Jean Tague, Ari Salminen and Charles McClellan. Complete formal model for information retrieval systems. In *Proceedings of the ACM/SIGIR International Conference on Research and Development in Information Retrieval,* 1991.

[26] James A. Thom, Alan J. Kent and Ron Sacks-Davis. TQL: Tutorial and user manual. Technical Report 92-19, Collaborative Information Technology Research Institute, 1992.

[27] Howard Turtle. Natural language vs. Boolean query evaluation: A comparison of retreival performance. In *Proceedings of the ACM/SIGIR International Conference on Research and Development in Information Retrieval,* page 212, Dublin, 1994.

[28] US Department of Defense. Military Specification. Markup Requirements and Generic Style Specification for Electronic Printed Output and Exchange of Text (SGML), 26 June 1993. MIL-M-28001B.

[29] E. Wilson. Electronic books: the automatic production of hypertext documents from existing printed sources. In *Fourth Annual Conference of the UW Centre for the New Oxford English Dictionary: Information in Text,* page 29, Ontario, Canada, October 1988.

[30] Eve Wilson. Converting an SGML text to hypertext. Technical Report TEI TR3 W6, University of Kent at Canterbury, 1991.

[31] Yaron Wolfsthal. Style control in the Quill document editing system. *Software - Practice and Experience,* Volume 21, page 625, 1991.