Amalgamating SGML Documents and Databases

Masatoshi Yoshikawa Osamu Ichikawa Shunsuke Uemura

Graduate School of Information Science
Nara Institute of Science and Technology

8916-5 Takayama, Ikoma, Nara 630-01, Japan
phone: +81 (7437)2-5331
fax: +81 (7437)2-5399
e-mail: {yosikawa,osamu-it,uemura}@is.aist-nara.ac.jp

Abstract

SGML is a markup language which can express informations on the logical structure of documents. By making use of these informations, documents can be retrieved not only by content-based but also by structure-based queries. In this paper, we propose a uniform and flexible mechanism to make reference links from SGML documents to database objects. In addition to typical document structures such as chapters, sections and paragraphs, our mechanism allows arbitrary words in documents as the source of these links. By using this mechanism, SGML attributes and their values of marked-up words can be transparently stored as database attributes, and we can establish hyperlinks between keywords in texts, which reflect the keywords' deep semantics. Also, we present a query language to retrieve SGML documents which are coupled with databases in this manner. The query language does not assume a particular database schema; instead, it utilizes DTD graphs, representing element structures of DTDs, as virtual schemas.

Keywords: Structured documents, SGML, hyperlink, query language

1 Introduction

SGML is an international standard document markup language[ISO86]. During the past few years, SGML has been gaining its popularity in wide area including (electronic) publishing, CALS and hypertext. One of the most popular application of SGML is the HTML used in WWW (World Wide Web). Two major components of an SGML document are a DTD (Document Type Definition) and a document instance. The DTD represents a grammatical rule of logical structure of documents, and the document instance is a document conform to the DTD. The document instance has, in addition to its contents, embedded tags to represent the logical structure and other attached information. A major goal of the SGML is to provide a framework to expedite the processing, exchange and reuse of texts by making use of the logical structure of texts. Since database technologies play a crucial role to attain this goal, SGML databases are emerging as an important research area[SDAMZ94, LZSD95, AMFL^{+}95].

Storing SGML documents by taking advantage of their logical structures makes it possible to retrieve texts not only by content-based but also by structure-based queries. Early studies in this direction include works of Macleod[Mac90, Mac91]. More recently, Christophides et. al. [CACS94, CR94] developed a query language, which is an extention of O_2SQL, to retrieve document instances. This query language assumes an O_2 schema which are enhanced to represent DTDs. Blake et. al. [BCK^{+}94] gave a framework to retrieve document instances by SQL utilizing virtual relations of which structure is independent of the DTD.

These past researches in this area are mainly focusing on the development of 1) database schemas to store SGML documents based on their logical structure, and 2) query languages to retrieve text components satisfying search conditions. Such development is an important step to achieve efficient storage and operation of large amount of SGML documents. However, since the SGML is highly versatile language, there exists a big potential of applying database technologies in other aspects of SGML document processing which include creation, editing and application. Databases play an important role to support the creation of high-quality texts.

As an approach toward this goal, we propose a fundamental mechanism to establish links between components of SGML documents, such as keywords and SGML *elements*, and database objects. The components of SGML documents here might be arbitrary character strings as well as document elements. Based on this fundamental link mechanism, we introduce a new logical layer on top of the SGML framework which enables a uniform representation of several references among SGML documents and databases. Furthermore, we present a query language which can express the retrieval of document elements based on both their logical structures (such as sections and paragraphs) and their contents.

More specifically, the main contribution of this paper can be summarized as follows:

1. We propose a uniform and flexible mechanism to make references from SGML documents to database objects. This mechanism allows not only document elements but also arbitrary character string in SGML documents as link sources. The reference mechanism, which realizes deep amalgamation between SGML documents and databases, yields the following advantages:

    - In SGML, logical constructs of documents such as sections, paragraphs and keywords can be "marked up" using tags. Tags may have attributes, called SGML attributes, which can be used to record some information on the marked-up blocks of character strings. With our mechanism, some of these SGML attributes and their values can be uniformly managed by databases.

- Using the information on keywords stored in databases, we can establish hyperlinks between those keywords in texts, which reflect the keywords' deep semantics. Both these hyperlinks and logical structure of texts can be used in queries.

  As far as we know, no other research in literature proposed such reference mechanism between SGML documents and database objects, which allows references from arbitrary character strings in texts.

2. The fundamental reference mechanism stated above adds new dimensions to the kinds of hyperlinks among documents and databases. We articulated the following three dimensions: i) links to document elements v.s. links to database objects, ii) marked-up elements as link source v.s. arbitrary character strings as link source, iii) direct links v.s. indirect links, Thus, the combination of these dimensions yields many kinds of links. We have newly introduced virtual element markup, which allows a uniform specification of these wide varieties of links in logical level.

3. Furthermore, we have developed a query language which can fully exploit the deep amalgamation of SGML documents and databases established by above mentioned linking mechanism. Our query language can express complex queries in which qualifications are specified by a combination of documents' logical structure, contents, links and database attributes. This query language can be used also to express indirect links mentioned in 2 iii) above. Another important feature of the query language is that it does not assume a particular database schema nor database model; instead, it utilizes DTD graphs, representing the logical structures of element in DTDs, as virtual schemas.

Furthermore, compatibility is another notable feature of our framework. Firstly, as for the above mentioned points 1 and 2, our mechanism is designed to be "upward compatible" to the SGML standard. Hence, existing SGML documents can be "upgraded" in our framework by making very minor modification to DTDs. Secondly, as mentioned in the point 3, since our query language does not assume particular database schema, our general idea can be applied to other schemas and query languages for structured documents proposed so far. Since our proposal is "orthogonal to" the trend of many other researches in this area, we expect the combination of our mechanism and other research results would yields synergistic effects.

In this paper, Section 2 gives an overview of the SGML. Section 3 describes a uniform framework amalgamating structured documents and databases. In Section 4, detailed discussions are given to establish links from arbitrary character strings in SGML documents to objects stored in databases. Section 5 illustrates a wide range of queries expressed using our query language. Section 6 presents both the static and dynamic reference mechanism between document elements. The dynamic references utilize the query language given in Section 5.

```
<!DOCTYPE article     [
<!ELEMENT article     -- (title, author+, abstract, section+)>
<!ATTLIST article     id    ID      #IMPLIED>
<!ELEMENT section     -- (title, p*, subsectn*)>
<!ATTLIST section     id    ID      #IMPLIED>
<!ELEMENT subsectn    -- (title, p+, subsectn*)>
<!ELEMENT title       -- (#PCDATA)>
<!ELEMENT author      -- (#PCDATA)>
<!ELEMENT abstract    -- (#PCDATA)>
<!ELEMENT p           -- (#PCDATA)>
<!ATTLIST p           idref IDREFS  #IMPLIED>
<!ENTITY  RDB         "Relational Database">
]>
```

Figure 1: An example DTD.

2 SGML

SGML (Standard General Markup Language)[Gol90, vH94] is an international standard markup language (ISO 8879[ISO86]), which allows the description of structures and attributes of documents. Basically, SGML documents consist of two components: DTDs and document instances. DTD (Document Type Definition) defines the grammar to which the logical structure of documents must conform. Document instances[1] , which has logical structure conform to the corresponding DTD. DTDs are defined for each type of documents, such as doctoral thesis, technical report, manual, and so on. In general, many document instances are created according to the logical structure defined by a DTD; hence, DTDs have some analogy with database schema, and document instances with database instances.

We illustrate these components using examples in more detail.

- DTD (Figure 1)

  A DTD consists of the following three parts:

  - **Element declaration** which declares the element names to form document instances, and define the logical structure, element name, occurrence order of the elements, etc. of documents.
  - **Attribute list declaration** which defines, for each element, SGML attributes, their declared values (or “domain” in database terminology), the default values and whether the attribute is mandatory or not.

---

[1] Document instance is a terminology used in the ISO standard. In literature, terms “marked document” or “marked text” are also used.

```
<!DOCTYPE article SYSTEM "article.dtd">
<article><title>Structured Documents</title>
<author> ... </author><author> ... </author>
<abstract> ... </abstract>
<section><title>Introduction</title>
<p>SGML is the language based on the ... </p>
</section>
<section><title>Basic Concepts</title>
<subsectn><title>General Markup Languages</title>
<p>The first general markup language is ... </p>
<p>In 1986, ISO ... <p>
...
</subsectn></section>
</article>
```

Figure 2: An example document instance.

  - **Entity declaration** which defines entities which can be used in documents and are replaced by SGML parsers into another predefined strings.

- document instance (Figure 2)

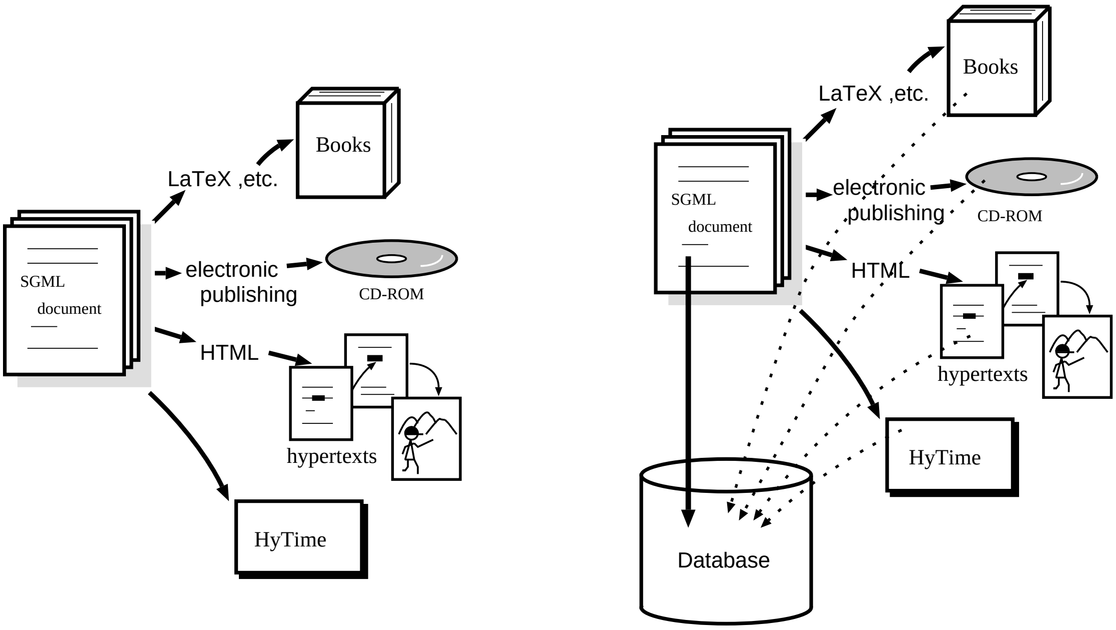
  A document, conforming to the corresponding DTD, embedded with various types of tags and entity references. Tags are used to specify the starting and ending positions of each logical structure such as chapters and sections. Moreover, by using entity references, systems will replace short reference names by long predefined strings.

In document instances, SGML *attributes* are described within the start-tags. The usage of SGML attributes is unrestricted. We can roughly classify the usage of SGML attributes as follows:

1. SGML attributes with values dependent on the contexts. The values are dependent on the tag occurrence which contain the attribute. For example, `figid` in `<figure figid='piechart1'>` uniquely designates a specific occurrence of element. Also, `type` in `<div1 type='section'>` designates the type of a specific element.

2. SGML attributes with values independent of the contexts. The values are independent of the tag occurrence which contains the attribute. `age` in `<person age=20>Joe Doe</person>` falls into this category.

In the SGML, *entities* are used for referencing to long sentences and strings by a short name, and for taking in documents stored in another file . In general, using *entities* makes it possible



(a) SGML document and applications.

(b) Reference mechanism from SGML documents to database objects.

Figure 3: Enhancing the SGML functionality.

- to refer to the words, which is an another document or (long) strings, which appear many times in the document with a short reference name, and to take them in an arbitrary place;
- to manage the declared contents of entities collectively; and
- to be referred from multiple DTDs using parameter entity as to the entities used in common.

3 A Uniform Reference Mechanism Amalgamating Structured Documents and Databases

Although the SGML provides a general framework for logical structuring of documents, the final usage of document instances conforming to SGML is out of the scope of the standard, and are up to application programs. In other words, SGML documents are "half-finished products". By using application programs for each purpose, we can obtain "final products" in a form of such as books, CD-ROMs, hypertext and etc. (Figure 3(a))

This implies that by enhancing the functionality of SGML documents as "half-finished products", we can build an environment in which every "final product" can enjoy the enhanced functionality. Our premise is that one important enhancement is to provide a general mechanism to make references from arbitrary document elements and character strings in SGML documents to database[2] objects (see Figure 3(b)). Such facilities are not supported within the framework of the SGML. This functionality of references yields the following advantages:

- SGML attributes of which values are independent of the position of the tag's occurrence (i.e. context) can be stored in databases, and can enjoy its advantages (i.e. easy maintenance of value update, concurrency control, indexing and etc.). A mechanism can be provided to make these database attributes be viewed as if they were ordinary SGML attributes. Also, schema evolution in the underlying database is reflected to the SGML attributes without changing DTDs.
- By marking up important words which represent real-world entities, and by using our reference mechanism, (a part of) semantics of words can be managed in databases. This makes it possible to utilize the informations of both logical structure and semantics of documents; thus
  - (indirect) hyperlinks, based on deep semantics, between words can be established; and
  - utilizing these hyperlinks, a wide variety of queries can be posed.

The introduction of this fundamental reference mechanism enable us to establish several kinds of linksbetween documents and databases. We classify these links along with the following three dimensions:

1. links to document elements v.s. links to database objects
2. macroscopic link v.s. microscopic link
3. direct links v.s. indirect links

In the 2nd dimension above, "macroscopic" means the link is established for document elements declared in DTDs. These document elements typically correspond to a block of sentences such as chapters and sections. Whereas, "microscopic" implies links for more atomic units of texts such as words and phrases. This paper takes into account microscopic markups besides macroscopic markups which many other studies have been focused on. As for the 3rd dimension, SGML provides direct links using `ID` and `IDREF`, but not indirect links. Since indirect links are indispensable to establish flexible linking, we will provide

---

[2] In general, SGML documents might also be stored in the "Database" in Figure 3(b). Although our discussion in this paper applies to such situations as well, we assume, without loss of generality, that the "Database" does not store SGML documents.

indirect links using our query language. The combination of the first and the third dimensions yields four links in all. Figure 4 illustrates these four links[3]. In the next section, we will focus our attention on microscopic links to database objects, and will discuss in detail the mechanism to uniformly specify the links.

4 Microscopic Links to Database Objects

As presented in Section 3, there are, in general, a wide variety of links. Among these many links, microscopic links referencing to database objects are not extensively studied in literature so far. In this section, we focus our attention on this type of links, and will illustrate the general framework to uniformly specify the links by using this type of links as example.

4.1 Using Attributes within the Framework of SGML

In this subsection, we describe the method to express the properties of real-world entities using SGML attributes, and reveal the problem of this method.

Let us consider representing important words and phrases in documents. For example, some properties of a person named "Doe" can be described using SGML attributes as:

`<person ssn=1234567 age=20 address="Nara">Doe</person>` (1)

Similarly, the population of the city "Nara" can be recorded as follows:

`<city population=240000>Nara</city>` (2)

Of course, specifying the pairs of SGML attribute names and values for each occurrence of the tagged contents "Doe" and "Nara" in structured documents will be a cumbersome task. The entity declaration below, which is part of a DTD, will ease the task. With this declaration, each occurrence of `&joedoe;` and `&nara;` in document instances is replaced by the above character strings (1) and (2), respectively.

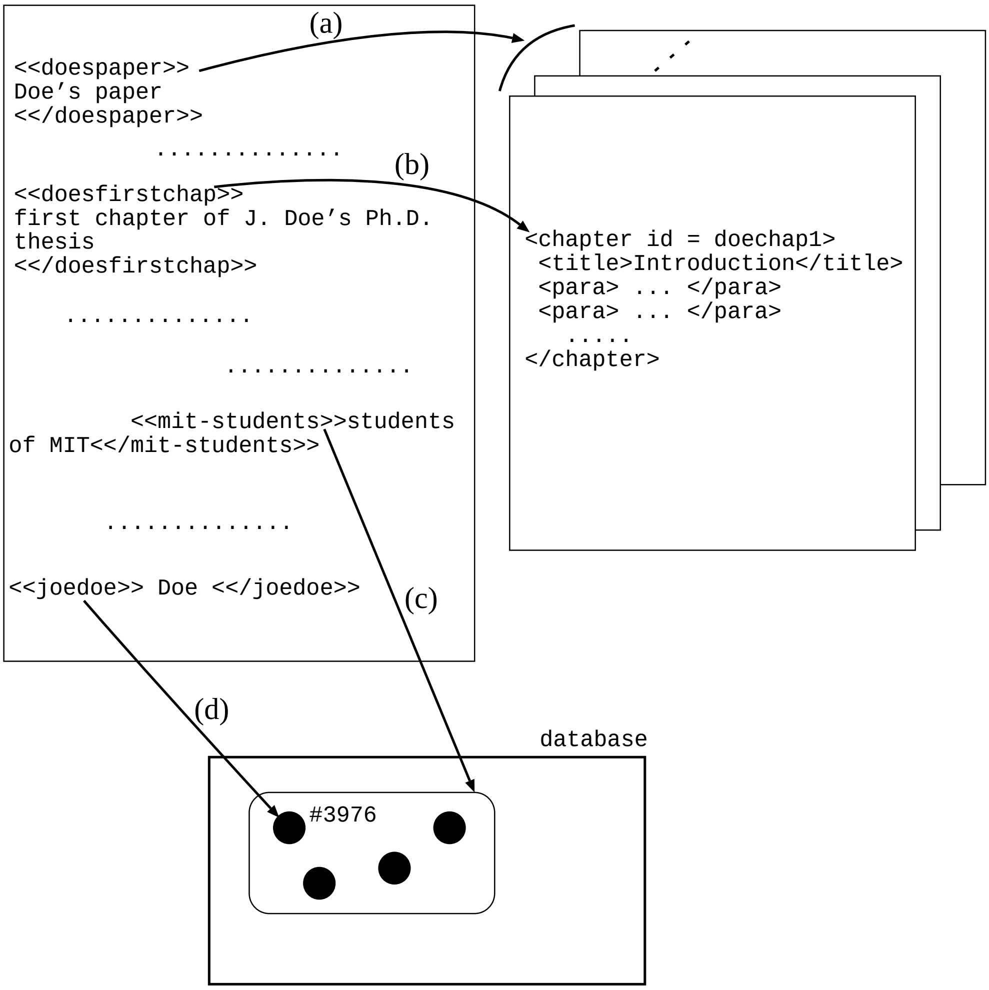
```
<!ENTITY joedoe "<person ssn=1234567 age=20 address="Nara">Doe</person>">
    <!ENTITY nara "<city population=240000>Nara</city>">
```

However, the management of attribute values within the SGML framework described above has the following problems:

Problem 1: The update of SGML attribute value(s) cause the rewriting of entity declaration in DTD, and re-parsing of every document instance which refers the markup containing the updated attribute(s). This scenario is, of course, unrealistic.

---

[3] All of these four links can be viewed as microscopic links.



dictionary

(a)	doespaper	indirect	**select** a.id **from** <article> a **where** a.<author> = "Joe Doe"
(b)	doesfirstchap	direct	doechap1
(c)	mit-students	indirect	**select** a **from** a **in** Student **where** a.university = "MIT"
(d)	joedoe	direct	**#**3976

Figure 4: Various links among structured documents and database objects.

Problem 2: In SGML, tag names and their attributes must be declared in DTD before document instances are created. However, tag names and their attributes in element markups (1) and (2) has exactly same semantics as classes and attributes of object-oriented databases. Hence, it is quite unlikely to be able to predict all classes and attributes of real-world objects referred in (unlimited number of) document instances.

4.2 Referencing Database Objects Using Generalized Tags

The SGML attribute values in element markups (1) and (2) do not depend on the occurrence position of tags which include those attributes. Therefore, a natural solution to resolve the Problems 1 and 2 in Subsection 4.1 is i) to store these SGML attribute values and tag names such as `person` and `city` in databases; and ii) to provide a mechanism to refer from tags in document instances to the corresponding database objects. To resolve the Problem 2, we need to use a single tag, say "`dbobject`", for referencing database objects. Using this tag, element markups (1) and (2) can be represented as follows:

`<dbobject type = person  oid = 1234>Doe</dbobject>` (3)

`<dbobject type = city  oid = 2345>Nara</dbobject>` (4)

The values of the SGML attribute `oid` in the element markups (3) and (4) are OIDs of "Doe" and "Nara" in the underlying database, respectively. Let us assume the attributes of the `person` type in the underlying database be `ssn, age` and `address`. By preparing an SGML application program which, given the element markup (3), generates the SQL sentence "`SELECT * FROM person WHERE oid=1234`", we can virtually provide users with the information of (1) in texts. By providing query results in this way, each `dbobject` elements can possess the attribute values stored in databases as if they were defined as SGML attributes. In this way, we can refer to any types of database objects by using the single generalized tag `dbobject` in a uniform manner.

Again in this case, using the element management facility by SGML entities described in Subsection 4.1, we can refer long character strings like (3) and (4) by short reference names. For this purpose, each `dbobject` element is declared in entity declaration; for example the element mark up (3) might be declared as:

```
<!ENTITY joedoe "<dbobject type = person  oid = 1234>Doe</dbobject>">
```

With this entity declaration, each occurrence of `&joedoe;` (i.e. entity reference) in document instance is replaced by the character string (3).

As stated earlier, we assume that SGML documents, which are conform to our proposing framework, are created from scratch. However, upgrading existing SGML documents to make them able to refer database objects is also very easy. The required modifications to DTDs are the following very simple one:

1. Insert the element and attribute declarations of the newly introduced tag `dbobject`.

```
<!ELEMENT root --
          ( … , …   ) +(dbobject)>
            …
<!ELEMENT dbobject -- (#PCDATA)>
<!ATTLIST dbobject type CDATA #REQUIRED
                   oid  CDATA #IMPLIED>
```

Figure 5: The DTD to refer to database objects.

2. Define `dbobject` in the root element of DTDs using the "inclusion" facility of SGML (`+(dbobject)` in Figure 5). By defining as an "inclusion", `dbobject` element can appear in any place within document instances conforming to the DTD.

The DTD after these modification is illustrated in Figure 5. Note that these modification does not affect existing document instances at all unless we do not utilize the added functionality of referencing database objects. Otherwise, if we want to use this functionality, the modification to document instances is just to insert `dbobject` tags to appropriate positions.

4.3 Virtual Element Markup

In the previous subsection, we have described basic machanisms of microscopic markups and the reference to database objects. To make this mechanism more versatile and easy to use, the mechanism still has the following problems:

1. Parameterization of taged contens (i.e. character strings.)

   The same real-world object is often denoted by several different character strings depending on their occuurrence positions. For example, a person may be denoted as "Doe" in some position and as "J. Doe" in another position within the same text. To deal with this situation with the method described in the previous section, we have to declare as many entities as different kinds of character strings. (The entity reference of the SGML does not provide a facility to parameterize a part of replaced character strings.)

2. The functionalities within the ordinary SGML framework should not be sacrificed. Specifically, we should be able to declare the SGML attributes whose values are dependent on the occurrence of tags.

3. The information on the linking to database objects should be shared among SGML documents.

In this section, we will introduce the notion of virtual entity markup to cope with these problems.

Firstly, we cannot meet the requirement 1 mentioned above within SGML framework because the entity reference of SGML does not have the functionality of parameterizing character strings. It is also difficult to meet the requirement 2 by simply extending the mechanism presented in the last subsection within the SGML framework. Thus, we will introduce a new logical layer on top of the SGML framework. Users logically create and view SGML documents in this new layer, and then those SGML documents are translated to fit into the ordinary SGML framework.

The notation of markups in this new logical layer is similar to ordinary element markup; "`<<`" and "`>>`" are used for tag open and close delimiters instead of "`<`" and "`>`". The general notation is as follows:

$$\texttt{<<ven } \mathrm{a}_1 = \mathrm{v}_1 \ldots \mathrm{a}_n = \mathrm{v}_n \texttt{>>A<</ven>>}$$

where `ven` is a *virtaul element name*, and $\mathrm{a}_1, \ldots, \mathrm{a}_n$ are SGML attribute names of `ven` whose values are dependent on the occurrence position of the tag (Hence these attributes cannot be stored in the underlying database). $\mathrm{v}_1, \ldots, \mathrm{v}_n$ are values of $\mathrm{a}_1, \ldots, \mathrm{a}_n$. Also, `A` is a content (i.e. character strings which appear in texts). By enclosing contents with a pair of start-tag and end-tag in this way, we can nest virtual element markup as in the case of element markups in ordinary SGML.

For example, let us assume that we want to use "Doe" to represent, in one context, a person having oid `1234`, and to use "J. Doe" in another context. Then, these representations are denoted as (5) and (6), respectively.

`<<joedoe>>Doe<</joedoe>>` (5)

`<<joedoe>>J. Doe<</joedoe>>` (6)

Furthermore, if we want to specify an SGML attribute only for (5), we can use the following notation.

```
<<joedoe comment = "...">>Doe<</joedoe>>
```

There are, in general, more than one virtual element markups having different contents but referring to the same database object (e.g. (5) and (6)). Since the information necessary to refer to the corresponding database object is common to all of these virtual element markups, by storing these information in a separate dictionary file, a preprocessor can translate, by looking up the dictionary, text files[4] containing the above mentioned virtual element markups into document instances (see Figure 6).

The extention described in this subsection requires the attribute list declaration given in the last two lines of the DTD in Figure 5 be changed as follows:

```
<!ATTLIST dbobject type    CDATA #REQUIRED
                   oid     CDATA #IMPLIED
                   sgmlatt CDATA #IMPLIED>
```

---

[4] We call these files "raw document instances".

dictionary

ven	type	oid
joedoe	person	1234
⋮	⋮	⋮

raw document instance

```
<<joedoe>>Doe<</joedoe>>
<<joedoe>>J. Doe<</joedoe>>
<<joedoe>>Joe Doe<</joedoe>>
```

DTD

```
<!ELEMENT root --
          ( … , …  ) +(dbobject)>
            …
<!ELEMENT dbobject -- (#PCDATA)>
<!ATTLIST dbobject type    CDATA #REQUIRED
                   oid     CDATA #IMPLIED
                   sgmlatt CDATA #IMPLIED>
            …
```

document instance

```
<dbobject type = person  oid = 1234>Doe</dbobject>
<dbobject type = person  oid = 1234>J. Doe</dbobject>
<dbobject type = person  oid = 1234>Joe Doe</dbobject>
```

Figure 6: The concept of virtual element markup.

5 The Query Language

By utilizing logical structural informations, SGML documents can be retrieved not only by content-based but also by structure-based queries. In order to make queries efficiently using logical structures of documents which DTDs represent, we introduce DTD graphs. A DTD graph is a directed graph which represents each elements in DTDs as corresponding nodes in its graph (see Figure 7). The query language utilizes DTD graphs as virtual schemas of documents.

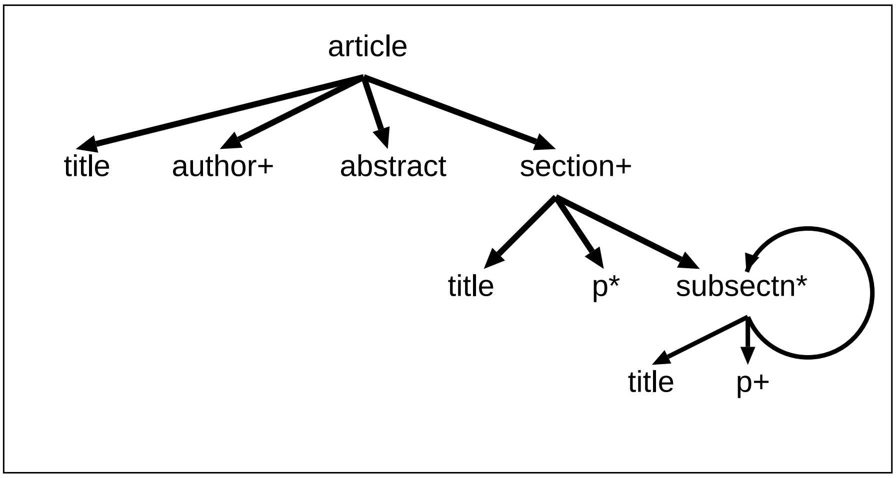


Figure 7: An example DTD graph.

Besides querying logical structures of SGML documents, with the amalgamation mechanism presented in the previous section, we can also retrieve documents on the basis of the information stored in underlying databases. The query language introduced in this section makes it possible to query the informations stored in databases which are amalgamated with structured documents in the same way as querying structured documents. The query language can also navigate hyperlinks represented by ID/IDREF. We will illustrate examples of queries which exploit deep amalgamation of structured documents and databases.

5.1 Queries on Logical Structures of Documents

5.1.1 Content-based Query

Q 1 *Find the title and the first author of articles having the abstract containing "SGML" and "HyTime".*

```
SELECT  <title>, <author>[1]
FROM    <article>
WHERE   <abstract> contains
         ``SGML'' and ``HyTime''
```

We specify element names having $STAGO$(`<`) and $ETAGO$(`>`). In the `FROM` clause, we specify element names defined in DTDs, and restrict the searched range. In the `SELECT` clause and the `WHERE` clause we should specify the lower elements than those specified in the `FROM` clause.

Elements defined by occurrence indicator("+" or "$*$") are specified by using "[ ]" notation in the query language. If we specify the `<`*element*`>`[$-n$], that means the times of the element from the last repetition of its element. Furthermore, the `contains` predicate is used to identify the element that contains the query terms.

5.1.2 Path Expression

Q 2 *Find the title and the abstract of articles containing "OODBMS" in the third paragraph of the second subsection of the first section.*

```
SELECT <title> , <abstract>
FROM   <article>
WHERE  <section>[1].<subsectn>[2].<p>[3]
                     contains  ``OODBMS''
```

In Q2, element names is specified by using dot notation to represent the parent-child relationship. The path expression `<section>[1].<subsectn>[2].<p>[3]` is a path selecting the first section, the second subsection of this section, and the third paragraph of this subsection. We can query paths like standard data.

5.1.3 Incomplete Path Expression

Q 3 *Find all titles in articles.*

```
SELECT   a..<title>
FROM     <article> a
```

The result of this query is a set of titles reached from the root of *article*. Such as Q3, we can query documents without exact knowledge of their logical structure by using ".." notation.

5.2 Queries on Database Objects through SGML Attributes

By amalgamating structured documents and databases, we can perform the various queries to documents using informations which are stored in databases.

5.2.1 Queries using Database Information

Q 4 *Find the title, the author and the authors' age of articles having an abstract containing "database".*

```
SELECT <title>, <author>, <author>.person.age
FROM   <article>
WHERE  <abstract> contains ''database''
```

In Q4, we assume that the element `<author>` has additional informations stored in databases. So we can retrieve informations stored in databases through documents by specifying database informations right on the heels of the element name.

Q 5 *Find large cities appeared in articles, and title.*

```
SELECT  <title>, a..<*>.city.name
FROM    <article> a
WHERE   city.population > 500,000
```

The `<*>` is the notation that represents all the elements, and we can also use ".." notation to represent an ambiguous path.

In the `SELECT` clause, we need the mechanism to find out the objects, which exist in the document, amalgamated with database "city". In the `WHERE` clause, the objects, which exits in the document, may exist throughout the document. But data concerned with the cities are just stored in the database "city", so we may just retrieve the database amalgamated with the documents.

6 Link Mechanism among Documents

In Section 4, we have proposed a fundamental mechanism to uniformly make a reference from arbitrary character strings in SGML documents to database objects. In this section, we turn our attention to hyperlinks between SGML document elements. First, we briefly presents static links which utilize the cross-referencing mechanism of the ID/IDREF of SGML. Then, we illustrate dynamic links which uses query language presented in Section 5. The destinations of dynamic links presented here may change depending on the information of the underlying databases as well as document structures and document contents. Finally, we will extend the virtual element markup to include the links to document elements.

6.1 Static Link Using ID/IDREF

In the SGML world, hypertext links between document elements are made using ID/IDREF functionality. SGML attributes which have the declared value ID are used to give a unique identifier to each element. These identifiers are referred from SGML attributes with declared

```
<!ATTLIST dbobject  type    CDATA  #REQUIRED
                    oid     CDATA  #IMPLIED
                    id      ID     #IMPLIED
                    idref   IDREFS #IMPLIED
                    dids    CDATA  #IMPLIED
                    sgmlatt CDATA  #IMPLIED>
```

Figure 8: Attribute declaration of the element `dbobject`.

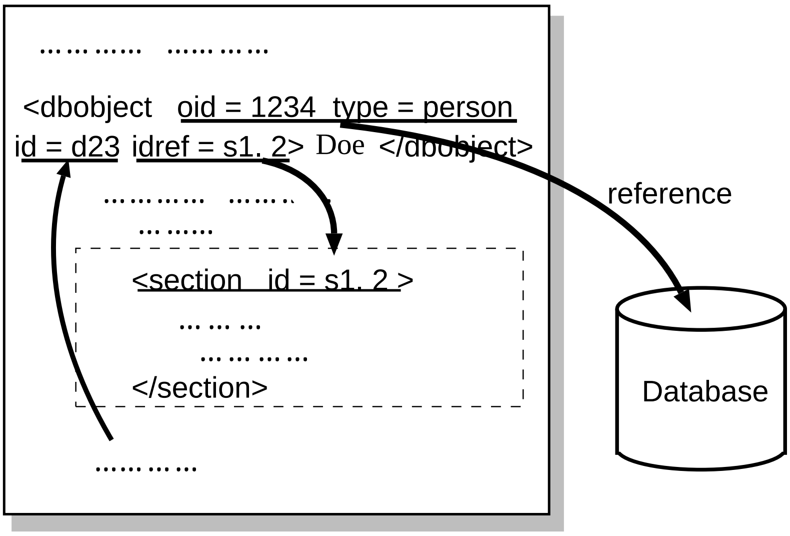


Figure 9: References using the element `dbobject`.

value IDREF. To make use of this functionality, all we have to do is just to declare SGML attributes with declared value ID (IDREF, respectively) in the attribute list declaration of the elements we would like to be referred (to refer, respectively). This is also true for the `dbobject` element introduced in Section 4. Hence, we extend the SGML attribute declaration shown in Figure 5. The new attribute list declaration after this extention[5] is shown in Figure 8. Note that we have declared the following two SGML attributes for the `dbobject` elements for reference purpose (see Figure 9):

- `oid`: for the references to database object.
- `id`: defined with the declared value ID, for hyperlinks.

---

[5] Figure 8 also includes the attribute `dids` which is explained in Subsection 6.2.

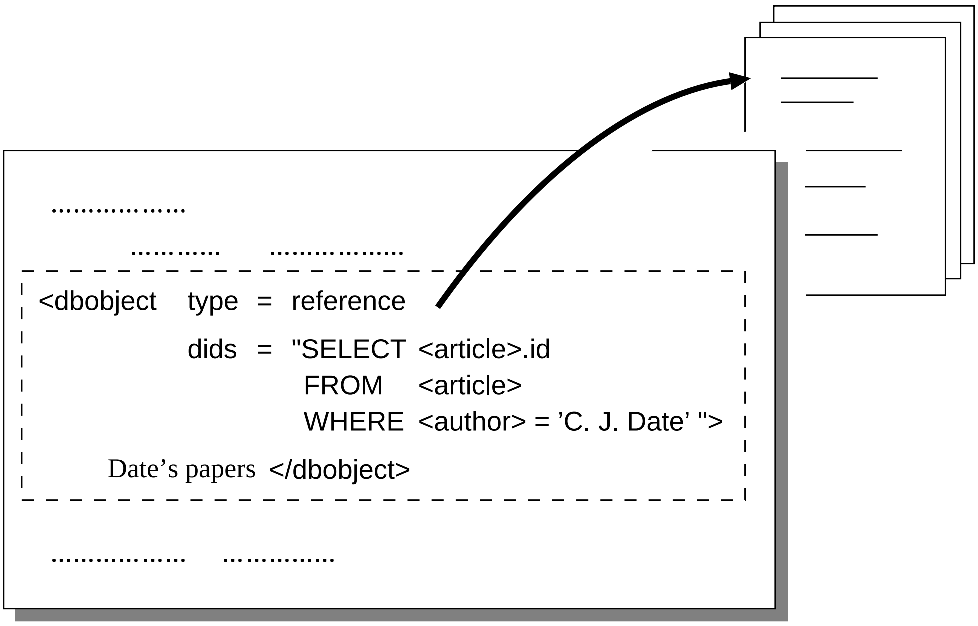


Figure 10: Dynamic link using `dbobject`.

The hyperlinks using ID/IDREF feature are static links whose destinations are fixed.

6.2 Dynamic Link

In the cross-referencing which make use of ID/IDREF, the referred elements are specified statically, and the cross-referencing is established by a unique identifier attribute. In this section, we describe the mechanism to decide the referred elements dynamically using the query language presented in Section 5.

An SGML attribute with IDREF, presented in the previous subsection, has an ID as its value. Whereas queries given in Section 5 returns a set of IDs as results. In general, the obtained set of IDs dynamically change depending on the set of document instances as well as the underlying database instance. Therefore, we can use queries to establish hyperlinks by using the IDs the queries return.

To hold query sentences in document elements for dynamic linking use, we need to define a new SGML attribute which has a query sentence as its value. We name the SGML attribute "`dids`", which stands for "dynamic IDs".

There are two types of queries to specify referred elements.

1. queries on documents (given in Subsection 5.1)
2. queries on both documents and underlying databases (given in Subsection 5.2)

We illustrate dynamic hyperlinks using these types of queries.

```
 …………
    <p   dids = “ SELECT <article>.id
                  FROM   <article>
                  WHERE  <abstract> contains ’multimedia’
                         and  <author>.person.age <= 30” >
        ……………
    </p>
…
```

Figure 11: Dynamic link using database informations.

Figure 10 illustrates a dynamic hyperlink using the first type of query. In this example, the character string “Date’s papers” is marked up with `dbobject` tags. The SGML attribute `dids` holds a query whose results may change depending on the current set of SGML documents. In this way, a dynamic link can be established from particular character strings in documents.

The next example illustrates the dynamic link using both document structures and the underlying database information. We can establish hyperlinks using information stored in the underlying databases which are amalgamated with elements in documents by specifying a query of type shown in Subsection 5.2 as the value of SGML attribute `dids`. For example, let us assume we would like to establish a hyperlink from a paragraph to the documents such that 1) the abstract contains the word “ multimedia” and 2) at least one of the authors’ age is less then 30, Note that the condition 1) is concerned with the document structure (and contents), whereas the condition 2) is concerned with the document structure and the information stored in the underlying database. Since the authors’ age changes year by year, the set of documents obtained as the destination of this hyperlink may dynamically change. Figure 11 represents the specification of this hyperlink. Again in this case, we can use the SGML attribute `dids` to hold a query sentence.

6.3 Virtual Element Markup for Document Hyperlink

Although dynamic links described in Subsection 6.2 are useful mechanism, specifying a long SQL sentence as shown in Figure 10 for each occurrence of the link in documents is a tedious task. Therefore, we will extend the notion of virtual element markup introduced in Subsection 4.3 to cover (both static and dynamic) hyperlinks between SGML documents. With the notion of virtual element markups and dictionaries, we can specify hyperlinks in documents with short virtual element names, from which a preprocessor can generate *entities* of `dbobject` elements.

As described in Subsection 4.3, the dictionary needs the following attributes to generate *entities*.

1. `ven` to specify contents of the object which exist in documents

`ven`	`type`	`oid`	`id`
tanaka	person	1234	s1.2
d-cars	car	select ...	–

date-papers	reference	–	`SELECT <article>.id`
			`FROM   <article>`
			`WHERE  <author> = 'C. J. Date'`

⋮	⋮	⋮	⋮

Figure 12: An example dictionary.

2. `type` of the object which the contents of documents represent
3. `oid` to refer to the object stored in databases

In Section 4, by using local dictionaries, *entities* are generated from virtual element markups, and it became possible to amalgamate with databases.

In this subsection, we will further introduce a new attribute in this dictionary, which represents elements' id referred from `dbobject` elements, in order to allow succinct specifications of hyperlinks using a virtual element markup in the same manner. An example dictionary is given in Figure 12. The attribute `oid` in this dictionary holds either an oid to statically specify a database object or an SQL sentence to dynamically specify oid(s). Whereas the newly introduced attribute `id` holds either an id to statically specify a document element referred from `dbobject` or an SQL sentence to dynamically specify id(s). Each attribute takes `null` value if there are no elements or objects to refer.

For example, if the above dictionary was defined, the dynamic link shown in Figure 10 can be specified succinctly as:

```
<<date-papers>>Date's papers<</date-papers>>                    (7)
```

The preprocessor replaces each occurrence of the virtual element markup (7) with the following `dbobject` element.

```
<dbobject  type = reference
           dids = SELECT <article>.id
                  FROM   <article>
                  WHERE  <author> = 'C. J. Date'>
          Date's papers  </dbobject>
```

Thus, by using the notion of virtual element markup and dictionaries, we can perform a succinct specification in the same uniform manner for the amalgamation with database objects as well as with document elements.

7 Conclusion

In this paper, we have introduced a new logical layer on top of the SGML architecture, which provides a uniform and flexible reference mechanism among SGML documents and database objects. We have also developed a query language which can fully exploit the amalgamation of SGML documents and databases. Although we illustrated, in Section 6, dynamic linking mechanism using our query language, the basic idea is independent of the query language used. Therefore, we can employ other query language (e.g. HyQ[Kim93]) to provide the dynamic linking functionality.

In this paper, we have assumed the underlying databases reside in a single site. However, in general, databases referred from within SGML documents may spread over wide-area network. In that scenario, we would need the introduction of the notions of "hosts" and "service" similar to those in CGI (Common Gateway Interface)[htt] of WWW. We are now implementing a prototype of our virtual element markup mechanism with the situation where databases being disseminated over the Internet.

References

[AMFL+95] T. Arnold-Moore, M. Fuller, B. Lowe, J. Thom, and R. Wilkinson. The ELF data model and SGQL query language for structured document databases. In *Proc. of 6th Australasian Database Conference (ADC'95)*, January 1995.

[BCK+94] G. E. Blake, M. P. Consens, P. Kilpeläinen, P. -Å. Larson, T. Snider, and F. W. Tompa. Text / Relational Database Management Systems: Harmonizing SQL and SGML. In *Proc. of First International Conference on Applications of Databases*, Vol. 819 of *Lecture Notes in Computer Science*, pp. 267–280. Springer-Verlag, June 1994.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 313–324, May 1994.

[CM95] Mariano P. Consens and Tova Milo. Algebras for Querying Text Regions. In *Proc. ACM Symp. on Principles of Database Systems*, May 1995. (to appear).

[CR94] V. Christophides and A. Rizk. Querying Structured Documents with Hypertext Links using OODBMS. In *Proc. of ACM European Conference on Hypermedia Technology (ECHT'94)*, pp. 186–197, September 1994.

[Gol90] Charles F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.

[htt] http://hoohoo.ncsa.uiuc.edu/cgi/.

[ISO86] ISO 8879: 1986. *Information Processing – Text and Office System – Standard Generalized Markup Language (SGML)*, Oct. 15 1986.

[Kim93] W. Eliot Kimber. HyTime and SGML: Understanding the HyTime HyQ Query Language. Available via anonymous ftp at ftp.ifi.uio.no/pub/SGML/HyTime, August 1993.

[LZSD95] Brian Lowe, Justin Zobel, and Ron Sacks-Davis. A Formal Model for Databases of Structured Text. In *Proc. of the 4th International Conference on Database Systems for Advanced Applications (DASFAA'95)*, April 1995. (to appear).

[Mac90] Ian A. Macleod. Storage and Retrieval of Structured Documents. *Information Processing & Management*, Vol. 26, No. 2, pp. 197–208, 1990.

[Mac91] I. A. Macleod. A Query Language for Retrieving Information from Hierarchic Text Structures. *The Computer Journal*, Vol. 34, No. 3, pp. 254–264, 1991.

[MAC95] T. Milo, S. Abiteboul, and S. Cluet. A Database Interface for Files Update. In *Proc. ACM SIGMOD International Conference on Management of Data*, May 1995. (to appear).

[SDAMZ94] Ron Sacks-Davis, Timothy Arnold-Moore, and Justin Zobel. Database Systems for Structured Documents. In *Proc. of the International Symposium on Advanced Database Technologies and Their Integration*, pp. 272–283, October 1994.

[vH94] Eric van Herwijnen. *Practical SGML.* Kluwer Academic, 2nd edition, 1994.