

A Semantic Network Approach to Semi-Structured Documents Repositories*

Vassilis Christophides, Martin Dörr, Irene Fundulaki

Institute of Computer Science
Foundation for Research and Technology - Hellas
Vassilika Vouton, P.O.Box 1385
GR 711 10 Heraklion, Crete, Greece
email: (christop,martin,fundul)@csi.forth.gr

Abstract. Using database technology for the administration of digital libraries offers many advantages in a multi-user and distributed environment. However, conventional DBMS are not particularly suited to manage semi-structured data with heterogeneous, irregular, evolving structures as in the case of SGML documents found in digital libraries. To overcome the difficulties imposed by the rigid schema of conventional systems, several schema-less approaches have been proposed. Using instead unconstrained, extensible schemata offered by object-oriented semantic network systems, we are able both to map document specific structures as database classes, and to model the associated constraint information as integrated schema annotations. In this paper we present the benefits of this approach to create, access and process heterogeneous SGML documents, and in particular to exploit the shared semantics of evolving SGML structures. A respective application is currently being implemented in the context of the AQUARELLE project.

1 Introduction

With the emergence of the World Wide Web (WWW), the amount, complexity and variety of digital libraries information currently available increase dramatically. Information collected in documents is handled by editors, navigators, search engines, etc. In most cases, the analysis of the sequential data stored in files is based on encoding formats and associated grammars. The Standard Generalized Mark-up Language (SGML) is becoming the prevailing standard [25] for digital documents creation, modification and exchange in a platform and system independent way. SGML encodes the structure of documents by marking their contents using tags according to Document Type Definitions (DTD). For instance, one of the most popular applications of SGML is HTML [32] used in WWW. It seems that the structuring of text has more explanatory than constraining character, a reason why HTML browsers e.g. are tolerant to unknown tags, and heterogeneity, polymorphism and irregularity becomes the

* Work partially supported by European TELEMATICS Project *AQUARELLE*.

rule for document structures (see for example cultural SGML documents DTDs [28, 35]). The distinction between data and structure seems to disappear, at least for the end-users: strict typing becomes counterproductive; complex and large schemata are created; queries and updates frequently address both, data and schema [3]. As it has been pointed out by many researches [42, 37, 14, 9, 10], the use of Database Management Systems (DBMS) for the administration of digital libraries offers many advantages: concurrent access and on-line updates, version management, declarative query languages, users authentication, recovery and protection of the stored information. However, conventional (relational or object-oriented) DBMS are not well suited to manipulate and exchange data with implicit, heterogeneous, irregular and rapidly evolving structure, as it is the case with semi-structured documents found in digital libraries.

During the last years, several schema-less approaches have been proposed [4, 12] for the management of semi-structured data in order to overcome the obstacles imposed by traditional rigid typing. We believe that omitting a priori a schema definition results in serious performance drawbacks and loss of functionality², even if the schema is given implicitly as in the case of SGML DTDs.

For this reason, we consider a more flexible approach based on object-oriented semantic network systems offering *unconstrained schemata*, which are *extensible at runtime*. With this approach, the structure of data a) can be defined a priori or dynamically created afterwards, b) is not strictly imposed and can easily be changed without reconstructing the whole database, and c) can be queried like normal data.

Furthermore, the capturing of document fragments in a database schema allows to integrate seamlessly the pure document storage with a wide range of applications associated with a document life-cycle, as collaborative authoring, configuration management and information reuse, workflow management, etc. [8]. As these applications are more and more distributed, heterogeneous SGML document interchange and integration becomes a major issue.

In this paper, we present a semi-structured document management system on top of the Semantic Index System (SIS) [24] that aims to cope with the above requirements. SIS is a persistent storage system, offering transaction management and concurrency control, which is based on an object-oriented semantic network data model [30]. We focus on SGML [25] documents, but our results can be applied as well to other compliant standards such as HyTime [26] or XML [2].

Our contribution is twofold: a) we define a conceptual model to manage SGML documents as semi-structured data in the SIS under an unconstrained schema (i.e. container classes), image of the associated DTD, which is annotated by the constraint or control information (i.e. processing data) of the corresponding SGML constructs in the DTD b) we show how this model can be used for automatic and semi-automatic merging of multiple DTDs into one SIS database by exploiting the shared semantics of equivalent SGML document fragments.

² At this point we must note the costly a-posteriori extraction of Data Guides [23] for easier query formulation and effective optimisation.

The powerful SIS viewing mechanisms can be used to query and update the content and the structure of SGML documents, which may be instances of arbitrary DTDs, and this at any level of detail the user needs.

The latter is very useful in order to import/export SGML document fragments together with the associated portion of the DTD. Even more, having an unconstrained schema for SGML documents, DTD variants as well as changes of the document structures can be incorporated on demand in the running system. In the same spirit irregularities of SGML document structures (i.e. SGML exceptions) can be easily captured by modifying the related schema on demand. For that sake, the classes, images in SIS of the corresponding SGML element declarations in the DTD, are created dynamically under the discipline of a fixed SGML specific metaschema. The underlying Semantic Index System (SIS) provides the required functionality. In applications requiring to enforce more specific SGML constraints, appropriate tools can use the processing data that is directly associated with the corresponding images of the SGML elements in the SIS.

The remainder of this paper is organised as follows. Related work is presented in section 2 outlining current strategies to map SGML to database constructs. A brief presentation of SGML is given in Section 3 by an example of a DTD for the description and documentation of museum collections. In Section 4 we describe the general knowledge representation mechanisms offered by the SIS-Telos data model. Section 5 introduces our model to map SGML DTDs and documents to SIS schemata and databases. In Section 6 we show how we can merge multiple DTDs into SIS according to our model and we analyse relevant application cases. A first implementation of an SGML to SIS loader is described in Section 7. Finally, conclusions and future work are given in Section 8.

2 Related Work

There is a growing interest in the development of text applications using DBMS technology. Since SGML provides a document exchange model rather than a general purpose data model [34], the use of SGML DTDs directly as database schemata [6, 29] is quite limited. The main question that arises is then, how SGML documents can be represented on a DBMS model in an optimal way. It has been illustrated by previous research work on this field, that the hierarchical structure of SGML documents can be more naturally represented using object-oriented models than relational ones [9]. Many recent studies focus on the representation of SGML documents on object-oriented DBMSs [14, 31, 10]. As practised so far, there exist two conflicting mapping strategies: a so-called “generic mapping” (i.e. minimally-structured) and a “DTD-specific” one (i.e. maximally-structured).

The former [18, 1, 21] uses one database schema (to represent a labeled tree) for all possible DTDs while the latter [14, 16] creates a new schema for each DTD. The advantages of the generic mapping are obvious: easier implementation of loaders, simpler manipulation of heterogeneous or irregular document structures as well as of multiple DTDs, genericity of the provided services. However, the

semantics of document structure is handled outside the database schema by compliant tools: editors, browsers, etc. Associated applications cannot refer directly to documents in the database with a specific meaning. For this reason, the DTD-specific mapping provides a one to one correspondence between SGML elements (or tags) with database classes (or objects), which allows for more sophisticated queries on document structure and content as well as for interesting optimisation techniques [15]. Unfortunately, the generation of a database schema for each DTD implicitly assumes that the structure of documents is static. Furthermore, capturing the data constraints imposed by the SGML constructs typically require considerable extensions to the underlying data model of the database system (e.g., introduction of union types).

We consider that the information contents of a DTD can be modelled by an *explanatory or informative part* defining container classes for the different document fragments (i.e. SGML tags) and a *constraint or control part* capturing structure constraints over these fragments (i.e. SGML constructs) in order to process documents inside or outside the database. The “semi-structured” mapping we present benefits from the expressive power and flexibility of data declarations in the SIS and combines the advantages of both approaches using a generic mapping at the metalevel (DTD independent) and a specific mapping (DTD dependent) at schema level. In addition, the annotation of the container classes by their corresponding SGML declarations in the DTD allows to share common grammatical rules for semantically equivalent document fragments after the merging of different DTDs or DTD variants.

Systems with similar motivations or goals have been presented in [31, 10], however the structure of SGML document fragments in SIS is derived directly from the declaration of elements in the DTD (unlike VODAK [10]) and the semi-structured characteristics of SGML documents (e.g. SGML exceptions) are naturally taken into account (which is not the case with the constraint-based model [31]). To our knowledge, generalised SGML repositories [20] offering data management at both, the DTD and the document level, have not been studied previously in the literature.

3 Basic SGML Constructs

Standard Generalized Mark-up Language (SGML) is an international standard [25] used widely in text applications such as digital libraries, electronic publishing, CALS, etc. SGML documents consist of two main parts: a Document Type Definition (DTD) and document instances. A DTD (see Figure 1) is essentially an extended context-free grammar [40] (i.e. with the connectors “,” “|”, “&” and the occurrence indicators “?”, “*”, “+”) declaring the *generic logical structure* of documents, while the *specific logical structure* is encoded in document instances by adding descriptive mark-up or tags. Tags are used to specify the starting and ending positions of the logical fragments of a document.

Figure 1 illustrates a DTD for the description of museum collections that will be used as a source for the examples in the remainder of this paper. It

```

    <!DOCTYPE Musemcol [
1. <!ELEMENT Musemcol - - (Musemobj)+>
2. <!ELEMENT Musemobj - 0 (Name, Photo?, Located, Artifact?)+(Bibliogr)>
3. <!ELEMENT Artifact - 0 (Creation, Modific*, Acquis?, Destruc?)>
4. <!ATTLIST Artifact shape (round | rectang | cube) rectang
    size (small | medium | large | huge) medium
    material CDATA #IMPLIED
    (weight | height | width | depth) NMTOKEN #IMPLIED>
5. <!ELEMENT (Creation | Modific | Destruc | Acquis) - 0 (Action)>
6. <!ELEMENT Action - 0 (Actor+ & Event)>
7. <!ELEMENT Actor - - (Person | Institut)>
8. <!ATTLIST Actor refactor IDREF #CONREF>
9. <!ELEMENT Person - 0 (Name, Place*, Born?, Died?)>
10.<!ELEMENT (Located | Born | Died) - 0 (Event)>
11.<!ELEMENT Institut - 0 (Name, Address?)>
12.<!ATTLIST (Person | Institut) id ID #IMPLIED>
13.<!ELEMENT Event - 0 (Time, Place)>
14.<!ELEMENT Place - 0 (Name+)>
15.<!ATTLIST Place id ID #IMPLIED
    replace IDREF #CONREF
    longit NMTOKEN #REQUIRED
    latit NMTOKEN #REQUIRED>
16.<!ELEMENT (Time | Name | Address) - 0 (#PCDATA)>
17.<!ELEMENT Bibliogr - 0 (#PCDATA) -(Bibliogr)>
18.<!ELEMENT Photo - 0 EMPTY>
19.<!ATTLIST Photo image ENTITY #REQUIRED>
20.<!ENTITY fig SYSTEM "/u/SIS/CLIO/image" gif>
21.<!NOTATION gif PUBLIC "+//ISBN 7923/NOTATION Graphic Interchange Format//EN">
]>

```

Fig. 1. A DTD for Museum Collection documents

uses concepts from the cultural information system CLIO [17] and represents a realistic though simplified view of museum information administered in form of SGML documents. We briefly comment the *Musemcol* DTD and present the basic SGML constructs [22] namely: elements (i.e. document logical fragments), attributes (i.e. additional mark-up information) and entities (i.e. references to content or mark-up data possibly stored outside the document).

A museum collection (line 1) consists of one or more (occurrence indicator “+”) museum objects (line 2). Each of them has (aggregation connector “,”) a name, an optional photo (occurrence indicator “?”), as well as data specifying the event of its relocation to its current place. The exception (+) in the content model of the element *Musemobj* allows to include bibliographic references everywhere within a museum collection document. Additional descriptions of the nature of a museum object, in the case it is an artifact (line 3), are the actions of creation, possible modifications (occurrence indicator “*”), acquisition, and destruction (line 5) as well as the physical dimensions of the object (line 4), given by the

SGML attributes shape, size, material, weight, height, width and depth. We consider an action (line 6) to be associated with at least one actor and an event, in any possible order (permutable aggregation connector “&”). An actor (line 7) may be a person or an institution (choice connector “[|]”). A person (line 9) has a name, an optional list of places, where she/he has lived, as well as an associated birth and death event (line 10). An institute (line 11) has also a name and optionally an address. An event in turn (line 13) is determined by a time and a place. A place (line 14) may have more than one (historical) names, as well as the longitude and latitude attributes (line 15) to denote its universal geographical co-ordinates. Time, name, address (line 16) and bibliographic references (line 17) are strings where the SGML exception (-) forbids nested bibliographic references.

Since the same historical places and persons (or cultural institutions) may be involved in the description of several museum objects in a collection, the SGML addressing mechanism is used to assign identifiers to document components (attributes of type ID lines 12, 15) and make references to these identifiers elsewhere in the document (attributes of type IDREF lines 8, 15). The value specification #CONREF for the attributes *refactor* (line 8) and *replace* (line 15) indicates that the corresponding elements actor (line 7) and place (line 14) have no content, when these attributes have values. Finally, SGML entities as *fig* (line 20) allow to include the associated photos (attribute *image* line 19) for museum objects and take into account their related notation (e.g., *gif* line 21) when the document is manipulated by an SGML editor or viewer.

We must note that the wide use in DTDs of the SGML element choice connector (“|”) and optional occurrence indicator (“?”) combined with the use of SGML exceptions and attributes specified as #CONREF, result in documents with a quite complex, polymorphic and irregular logical structure. These modeling primitives of SGML give the documents their semi-structured character.

4 A Brief Presentation of the SIS Data Model

The Semantic Index System (SIS) [24] is a persistent storage system based on the object-oriented semantic network data model Telos [30]. SIS-Telos data model [5] provides: *unbounded classification* hierarchy (metaschemata see schemata as data and allow to develop a schema under this specific discipline) *multiple instantiation* (one object-class can be instance of more than one classes using the attributes of all its classes-metaclasses), *multiple inheritance*, as well as *optional and multivalued attributes* which can also have their own attributes. Note that SIS attributes are first class citizens and they can have their own classification classes (or metaclasses) stated as *attribute categories* (or *metacategories*). SIS attributes represent something like a generalisation of the notion of fields and references in conventional DBMS.

The notion of a schema in SIS is *ultimately unconstrained*. It is only required, that the from- and to-value of any attribute are declared instances of the classes foreseen by the schema, directly or through inheritance. The query engine supports retrieval by multiple and recursive conditions, as well as navigation through

the entire network of semantic relations defined in our model by attribution, as well as the class-instance and class-subclass relations. The graphical user interface, with various customisable and extensible views, allows the visualisation of complex graph structures such as SGML DTDs and instances.

5 Mapping SGML DTDs to SIS schemata

In this section we define our conceptual model to represent SGML DTDs and instances in the Semantic Index System (SIS) [24]. We follow the idea, that DTDs can be interpreted as an analogue to database schemata, and documents to database instances. However, the major obstacle to establish this correspondence are the differences between the modeling primitives, especially data constraints (i.e. the SGML element connectors and occurrence indicators) used in a document exchange model, such as SGML, and a general purpose data model, such as SIS-Telos [30]. Rather than using the DTDs directly as database schemata or extending the underlying model of the DBMS to capture the data constraints imposed by the SGML constructs (see section 2), we consider an SIS-Telos model where the information contents of a DTD can be modelled by an explanatory or informative part defining container databases classes (images of the SGML elements in the DTD) for semi-structured documents fragments and a control or constraint part (corresponding to the SGML element declarations in the DTD) for the processing of the container classes instances, under an SGML point of view, inside or outside the database.

The explanatory and the control part of our model are defined in SIS as two tightly coupled conceptual metastructures. The former, stated in the sequel as *container model*, allows to define for each DTD an SIS schema where the declared classes and their attributes serve as DTD-specific explanations and semantic indices on the encountered instances. The container classes allow then to represent documents as trees of nodes characterised by attributes. The explanatory nature is supported by the fact, that SIS classes are not bound to one particular type (i.e. tuples) and SIS attributes are optional and repeatable by default. This implies, that any object (or attribute) can be classified under multiple classes (or categories) - in particular one schema may cover only parts of a graph of interconnected data. Due to that, the required flexibility of a "generic mapping" is maintained for heterogeneous or irregular documents, without losing the semantic support of a "DTD-specific" schema (see section 2).

The latter conceptual structure, stated in the sequel as *processing model*, allows to represent the different grammatical rules encountered in a DTD according to the SGML metagrammar, as a direct annotation on the classes of the container model, which represent the equivalent DTD items the rules refer to. These rule representations are not further instantiated, but serve as an information and configuration service to all processing components on top. This is enabled by the fact, that the SIS allows to attribute schema items with non-instantiable items. It must be stressed, that this direct linking between the container and the processing data is a definite advantage (and only there) on a system, where the

schema changes at run-time and the schema and the classification of data items can be queried. For applications requiring to enforce more specific SGML constraints, appropriate tools can use the processing data directly associated with the corresponding images of the SGML elements in SIS.

To conclude, annotating the document fragments stored in SIS via the explanatory model with the corresponding rules in the DTD offers a good compromise between a “generic” and a “specific” mapping with many advantages:

- uniform storage of SGML documents with heterogeneous or irregular logical structures and SGML processing data on document structure and content;
- merging of different DTDs (i.e. schema integration) or definition of DTD variants (i.e. schema evolution) by sharing common grammatical definitions;
- queries on DTDs/instances enabling document fragment exchange with the associated portions of the DTD;
- structured views on SGML documents, which may be instances of arbitrary DTDs.

In the sequel we detail the representation in our conceptual model of SGML DTDs. We focus on the container and processing aspects of the different SGML constructs namely elements and attributes both at the metaschema (DTD independent) and schema (DTD dependent) levels using as example the *Musemcol* DTD presented in section 3.

5.1 SGML Elements

SGML elements are used to identify the logical fragments of documents as well as to declare the structural relationships among them. Each element has a name and a content defined by an associated model (see element *Musemcol*, line 1 of figure 1). The left part (i.e. the container) of figure 2 represents the various SGML elements encountered in a DTD (metaclass **SGMLElement**) while the right (i.e. the processing) their corresponding SGML models (metaclass **ElementModel**). Instances of the metaclass **SGMLElement** are the container classes images of the DTD elements, as for example **Musemcol**, and instances of the **ElementModel** are the associated element models, as for example (**Musemobj**+). The attribute metacategory “*elmRule*” is defined to establish the relation between a container class and its corresponding SGML content model (i.e. a link between the container and the processing model). The container classes are instantiated for each element instance (tag) encountered in a document.

Since the logical structure of SGML documents is essentially a tree, we specialise **SGMLElement** to the metaclasses **Terminal** for the leaves of the tree and **NonTerminal** for its intermediate nodes. In our example, the class **Photo** is an instance of the former, while the class **Musemobj** is an instance of the latter. In the metaclass **NonTerminal** we define the attribute metacategory “*elmContent*” to refer to all structural relations of a container class to those classes defined in its content model. This means, that for each different class being referenced, a specific attribute is defined which allows to store and query the respective instances separately. This is one of the major differences to the “generic mapping”.

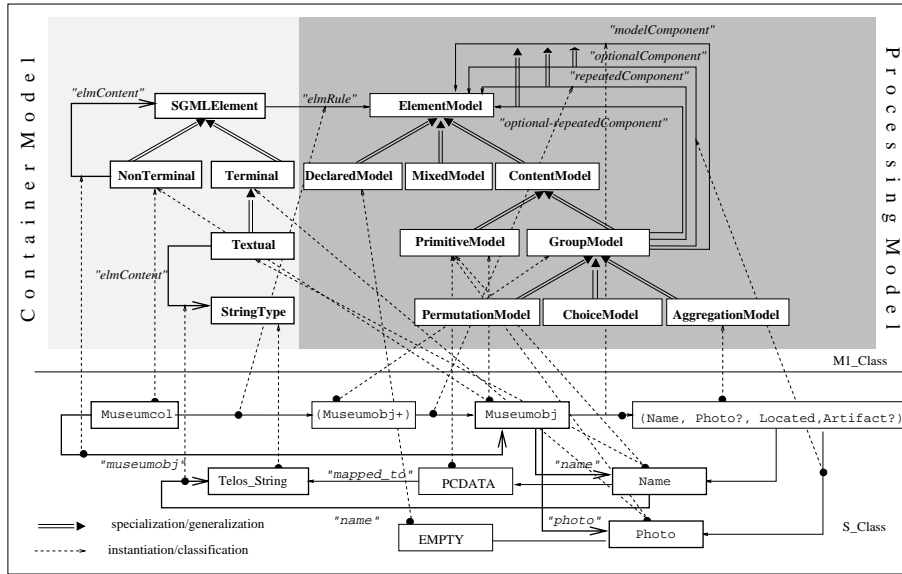


Fig. 2. The container and processing model for SGML elements

For instance, the class `Museumobj` is consequently linked with its contained classes `Name` and `Photo` through the instantiable attribute categories named “`name`” and “`photo`” (which in turn are instances of the metacategory “`elmContent`”). Note that the fact that all attributes are optional and repeatable per default in SIS obviates the introduction of null values and bulk data types (as in traditional ODBMS). Furthermore, SIS classes are not bound to one particular type. This is one of the major differences to the “DTD-specific mapping”.

Terminal elements can be either empty or textual. Therefore the metaclass `Terminal` is further specialised to `Textual`, where the metacategory “`elmContent`” refers a metaclass `StringType` with the only instance `Telos_String`, as for example in the case of the container class `Name`. This means, that textual terminal elements have an attribute which contains the actual string.

We consider now the representation of element models. An element model is a grammatical rule, either an SGML data type declaration for a textual (i.e. CDATA) terminal or empty (`EMPTY`) terminal element, that does not have content, or a more complex SGML declaration for a non-terminal. The specialisation of `ElementModel` to the metaclasses `DeclaredModel` and `ContentModel` is straightforward. In addition, the metaclass `MixedModel` represents element models (i.e. ANY) allowing a free mixture of textual data and any of the SGML elements defined in the DTD.

An element model is not further instantiated for documents, but it declares the rule according to which an attribute (link) of the metacategory “`elmContent`” was generated. Hence the container classes connected by such a link are connected by an element rule as well. More than one such link may be generated

by the same rule as in the example of (**Name**, **Photo?**, **Located**, **Artifact?**). But a link may also be generated by more than one rule, as will be later described in the context of DTD merging (see section 6). This was a major motivation to separate the container model from the processing model. The latter allows any processing tool to access the locally applicable rules directly from the respective element instances (as SIS allows to query the instance-class relation), without reparsing the whole document ahead. The idea is to store the rules in a form, which needs no further parsing for interpretation (the rule names in figure 2 are only mnemonics), and allows to represent any shared information as shared nodes and links in a minimal way. Consequently, we undertake the expansion of complex rules described below. Nodes (classes) are chosen for self-consistent constructs, and links (attributes) for those specifying different kinds of relations between nodes.

Complex SGML models are defined using other elements (or the `#PCDATA` data type for strings that need eventually further parsing) possibly combined with SGML connectors (i.e. “&”, “|”, “,”) to declare permutation, choice and aggregation of elements. Nested definitions of content models are also possible. The metaclasses **PrimitiveModel** and **GroupModel** as well its corresponding subclasses reflect this classification. The attribute metacategory “*modelComponent*” defines the relation of a **GroupModel**, as (**Museobj+**), with its (nested) fragments, as (**Museobj**). The fact that element names can appear both in the declaration of other element models and in their own content model, is captured through multiple instantiation under the corresponding metaclasses in the container and the processing model of our conceptual model. For example, the class (**Museobj**) is a common instance of the metaclasses **Terminal** and **PrimitiveModel**.

Since content models can be qualified by occurrence indicators (“?”, “+”, “*”), “*modelComponent*” is further specialised to the attribute metacategories “*optionalComponent*”, “*repeatedComponent*”, and “*optional-repeatedComponent*”. For example, the outgoing link of the (**Museobj+**) to its element class (**Museobj**) is an instance of the metacategory “*repeatedComponent*”. Attributes are used, because the indicator defines the way in which the model refers to the associated elements (i.e. kind of relation).

Finally, SGML defines textual primitive data types such as `#PCDATA`. The corresponding SIS-Telos primitive `Telos_String` has the correct container behavior, but not the SGML processing semantics. Hence SGML textual primitives are modelled as static part of the processing model and linked via the attribute category “*mapped_to*” to the SIS-Telos class `Telos_String`. The latter is referred in the container model and used for actual instantiation.

The issue of SGML Exceptions

Associated with the content model of an SGML element may be a list of exceptions. Exceptions allow to modify the content model of elements as well as of their subelements and can be viewed as a syntactic sugar of SGML leading to more human-readable forms of DTDs. More precisely, inclusions (see line 2

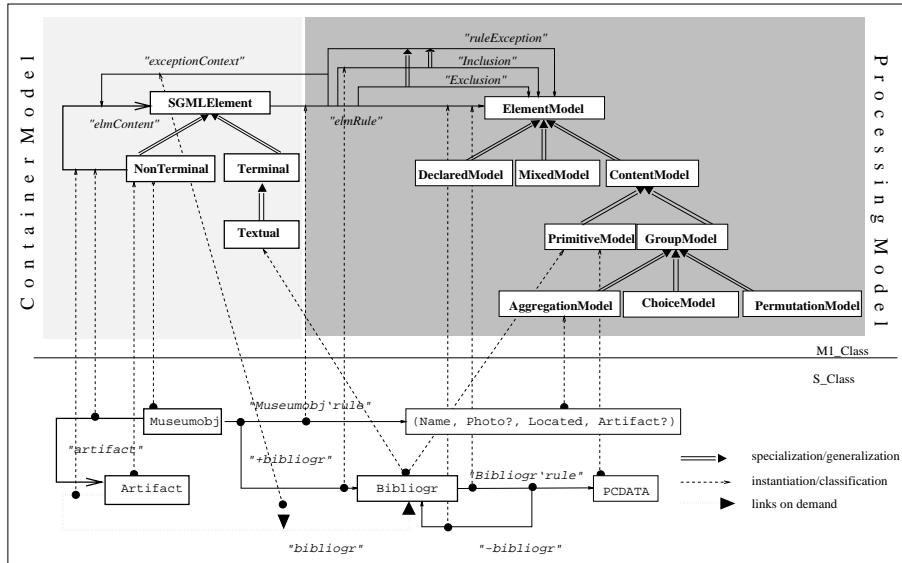


Fig. 3. The container and processing model for SGML element exceptions

of figure 1) allow one or more SGML elements (e.g., bibliographic references) to appear anywhere within the content of an element (e.g., *Museumobj*) or its subelements (e.g., *Artifact*), while exclusions (see line 17 of figure 1) preclude them from the element content (e.g., nested bibliographic references) and the from content of its subelements. Exceptions are one of the most salient features of SGML that gives to documents their semi-structured character (i.e. irregular structures). In order to store SGML documents in an DBMS, SGML exceptions are usually compiled (see “DTD-specific mapping” section 2) to produce an equivalent DTD without exceptions [27]. Then, this quite complex DTD is mapped to a DBMS schema. In our approach we represent the SGML exceptions encountered in a DTD only in the processing model (i.e. right part of figure 3) by the attribute metacategory “*ruleException*” while their effect on element structure is captured dynamically in the container model during the loading of the corresponding document instances.

Since an SGML exception essentially associates the model of an element with the corresponding included or excluded elements, “*ruleException*” is defined from the attribute metacategory “*elmRule*” to **ElementModel**. The different exception types are represented by specialising “*ruleException*” to the metacategories “*Inclusion*” and “*Exclusion*”. For example the link “*+bibliogr*” from “*Museumobj*’*rule*” to **Bibliogr** is an instance of the former while the link “*-bibliogr*” from “*Bibliogr*’*rule*” to **Bibliogr** is an instance of the latter. Such links are created during the analysis and the loading of the DTD.

We have seen that the effect of SGML exclusions in element structure is propagated from the elements where they are defined to their subelements. We must

note that different exceptions may be propagated to the same subelements, with possibly conflicting results, depending wherever they are involved (direct or indirect) in many element models. Capturing the propagation of SGML exclusions requires the introduction of a context notion in which the excepted elements are allowed or disallowed for a given subelement³. In our example, bibliographic references can appear in the instances of **Artifact** since they are defined in the context of its ancestor container class **Museumobj** but cannot appear in the instances themselves of **Bibliogr** which is also a descendant of **Museumobj**. For this reason we define in the container model the attribute metacategory “*exceptionContext*” from “*ruleException*” to “*elmContent*”, in this case an attribute from an link to a link. Instances of “*exceptionContext*” and “*elmContent*” are created on demand whenever a legal exception is encountered for an element during parsing of document instances i.e. we generate a schema entry together with the data item to be stored in.

For example, when parsing a valid instance of the *Museumcol* DTD, we encounter in the content of *Artifact* an element *Bibliogr* we first instantiate the metacategory “*elmContent*” by the link ‘**‘bibliogr’**’ from the class element **Artifact** to **Bibliogr**, and then create the instance of class **Artifact**. Since the SGML parser can also identify the element (i.e. *Museumobj*) from which the exception is propagated we can instantiate the metacategory “*exceptionContext*” by the attribute associating the links ‘**‘+bibliogr’**’ and ‘**‘bibliogr’**’.

This model allows to control the actual use of exceptions, e.g., if the compatibility of a large document with another DTD variant has to be determined (see section 6). Even though a processing tool will not always find the applicable exception locally, as above for normal content models, it has only to follow the document tree from the element with the exception up to the root, to find out all applicable inclusions/exclusions. This is still very efficient, as SIS provides a fast link traversal.

5.2 SGML Attributes

SGML attribute declarations are by far less complex, and our mapping follows the same principles as for elements, so we restrict ourselves here to the basic idea. SGML attributes relate an SGML element to an SGML attribute value under a specific name (see attribute *material* of element *Artifact*, line 4 of figure 1). We separate again the container model (see upper part of figure 4), which is further instantiated, and the processing model (see lower part of figure 4). The container aspect of this SGML construct is directly equivalent to the SIS-Telos attributes with the same name, represented by the metacategory “*elmAttribute*”. The processing aspect is captured by the metaclass **AttributeModel**. The attribute metacategory “*attRule*” establishes the association between both, in this case an attribute from an link to a node. Again the rule is expanded in its constituents (i.e. attribute value type, specifications and default value). As above, classes

³ A similar mechanism is used by existing SGML parsers [39] to interpret exceptions during the syntactic analysis of documents.

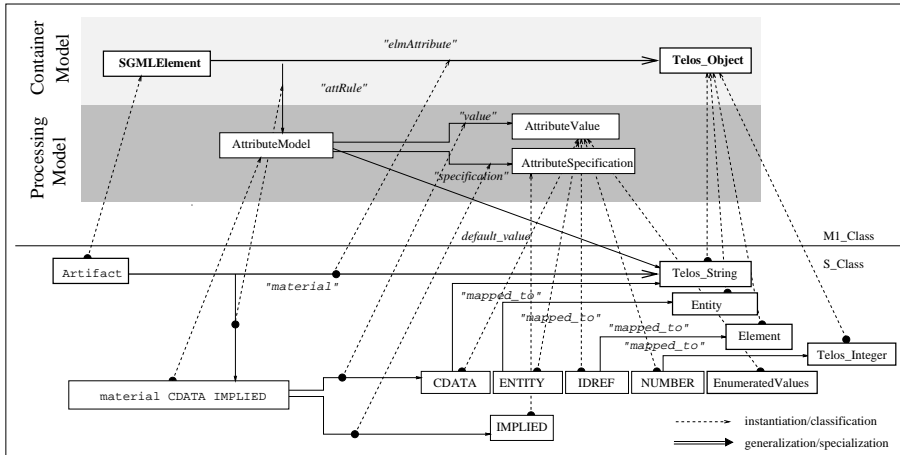


Fig. 4. The container and processing model for SGML attributes

are shared notions between the container and the processing model. The static mapping of the SGML element textual types to classes of the container model is extended here for the various SGML attribute value types.

There are two cases of SGML attribute value types requiring a special treatment. First the type `#IDREF` allows to refer to any element (see attributes *refactor* and *replace* lines 8 and 15 of figure 1) where an attribute of type ID has been defined (see attributes *id* lines 12, 15 of figure 1). SGML parsers do not interpret cross references and they usually consider them as strings. In our context these references should be interpreted when possible. For that sake, we foresee the class **Element** and all container classes with an attribute of type ID (instances of **SGMLElement**) will be declared as subclasses of it. Second the type `ENTITY` allows to refer to any entity defined in the context of the document DTD (see attribute *fig* lines 8 and 15 of figure 1). SGML entities essentially define macros or constants to be used in the framework of a DTD or its corresponding instances. We model them in the container model as one static schema, represented by the class **SGMLEntity**. When a DTD is parsed and mapped to the SIS, its entities are loaded as a pre-population of the data-level. These entities can be in the sequel used directly as to-values of the related attributes. This is particular useful to capture links of SGML documents to their subdocuments represented by SGML entities.

This kind of mapping for SGML attributes should be very advantageous for applications requiring a richer semantics of attribute types (and in particular of references) than currently provided by SGML, as discussed recently in the context of HyTime [26] or XML [2] initiatives.

6 Merging multiple SGML DTDs in SIS

In the previous section we have defined a conceptual model to store SGML documents in SIS as trees of nodes with attributes, annotated by their corresponding grammatical rules in the DTD. In this section, we show how this approach can be used for automatic and semi-automatic merging of multiple DTDs into one SIS database. Thus the system can store in parallel document instances of arbitrary DTDs, but it can also support advanced services for DTD comparison: detection of structural discrepancies, common document fragments, similar parts etc., and exploit this information for the respective handling of document instances.

The wish to handle multiple DTDs in one database originates in several application areas [7]. In the simplest case, the user just wants to maintain a digital library of heterogeneous documents regardless of their structure and origin, but nevertheless benefit from the advantages of an integration of the different DTDs in one repository schema (analogous of those in the “generic” approach, see section 2). This kind of DTD merging can be achieved automatically by the system based on a pure lexical and syntactical analysis of the different DTDs. As with database schemata however, we are frequently confronted with the problem that different DTDs share common semantics or describe the same “world” in different terms. We may distinguish two situations: either independent teams develop from scratch DTDs (or DTD parts) for a similar problem or application or a team modifies an existing DTD in order to cope with additional or changing requirements. The respective interpretation analysis and handling of different, seemingly identical and similar constructs is quite different.

In the former case we are interested in a mechanism to view all document instances under a kind of “global schema”, analogous to heterogeneous databases schema integration [33, 38, 11]. This approach allows access to heterogeneous documents manipulated locally under their own DTD, eventually integration of fragments coming from heterogeneous sources into a new document or even documents transformation from one DTD to the other. In general, each DTD has its own proper document instances. Between the DTDs, we do not expect *a priori* any element or attribute to be the same. Besides occasional linguistic similarities or structural compatibilities between elements or attributes having the same meaning, a stronger structural equivalence requires human assessment, and automatic tools can only support such decisions. We claim here that respective applications can advantageously be built on top of our conceptual model, as it provides the necessary analysis of the DTDs, and the access points and “hooks” within an extensible framework.

The latter case can be split into the slightly different notions of DTD versions and variants (or configurations) similar to the issue of databases schemata versioning and evolution [41, 36]. In the case of DTD versions we would like to enable efficiently the access to obsolete forms of document instances until they get out of use. In the case of variants, we would like to maintain simultaneously forms with shared and non-shared parts for specific target groups within a text application. Note that variants may also have versions. The goal is to optimise creation, access and maintenance of the shared parts. Typically, a sys-

tem can detect the shared parts of DTDs and related instances automatically, given that naming and structure was changed under a certain discipline known to the system. Tag names are typically thought to help the end-user by linguistic associations. This creates the need to maintain pure renaming variants in international applications as for instance in the AQUARELLE project⁴.

Central to the automatic shared part handling are the notions of equivalence of SGML element and attribute declarations, and the degree and granularity to which SGML constructs can be identified and maintained as shared or as specialisation or generalisation of each other. We consider equivalence of elements and attributes at three levels: a) Their naming (i.e. possible alias) b) Their structuring or value typing disregarding occurrence indicators or value specifications (i.e. only the container model) c) Their full content model (i.e. the container and the processing model). What remains, are non-equivalent SGML constructs we can deal with as variants (i.e. distinct container and processing models).

At first we consider only the SGML elements or attributes defined in their scope having the same name as potentially equivalent. If they share the full content model, we regard them as identical; if they share the generated container model, as compatible. Then, they are mapped to the same container classes or class properties, while in the case of identity they also share the same annotations for the associated grammatical rules.

In this kind of "trivial merge" fits also the case where two elements have structurally equivalent grammatical forms, as for instance, the following redefinition of the element *Action*: `<!ELEMENT Action - 0 (Event & Actor+)>` (see line 6 in the DTD of figure 1). In order to decide if two SGML grammatical forms are structurally equivalent we need a complete analysis and comparison of both content models based on their connectors and occurrence indicators. Similar algorithms are used by commercial SGML parsers [19] or DTD builders [13] to produce optimal internal representations of the DTDs.

In a more complex case, element structures can be recognised as pure extensions of another one, i.e. as specialisation. Their representation of grammatical rules are declared as subclasses of each other allowing again for transparent access to the shared properties. Consider, for instance, a redefinition of the element *Institut* as follows: `<!ELEMENT Institut - 0 (Name, Address?, Director)>` (line 11 in the DTD of figure 1).

In order to merge the new declarations of the elements with the old ones we factorise the common subparts of grammatical rules as an inheritance case, illustrated in figure 5. In the same spirit we can merge compatible declarations of SGML attributes as for instance, the new declaration of the attribute *material*: `<!ATTLIST Artifact material NAME #IMPLIED>` (line 4 in the DTD of figure 1).

As we can see in figure 6 we consider the attribute model `NAME` (kind of SGML string type) as a special case of the model `CDATA`. For such type of equivalence preserving merge of DTDs we need to establish compatibility rules for the different SGML (element or attribute) data types (e.g., between `CDATA`

⁴ <http://aqua.inria.fr/>

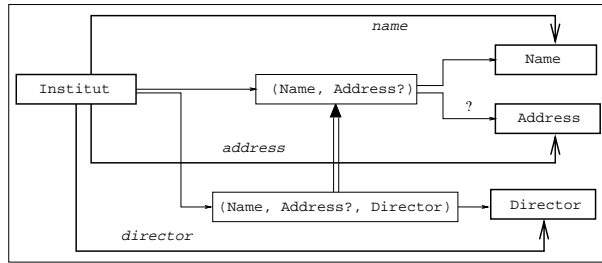


Fig. 5. An example of SGML elements merging to SIS

and NAME) and to compare recursively complex content models based on their connectors and occurrence indicators. A complete presentation of a merging algorithm for compatible content models is beyond the scope of this paper. Note that content models analysis can be naturally achieved within the framework of our conceptual model.

If the full container models fit, but not the names, the constructs are regarded as isomorphic. In this case, the desired identity or compatibility can be re-established by the use of alias names specific to the respective DTD. In general, this needs human control, as many trivial cases can be thought of, where some structures accidentally fit. Only if the history of the difference is known, it can be automated, as e.g. if someone generated a new version of a DTD by only changing names. Afterwards, the above analysis of identity and compatibility is reapplied.

Finally, if the names agree, but not the content models there is a conflict in either of the above forms. In this case, the constructs are mapped to different class elements or class element properties, and an alias is applied to restore the appropriate distinct naming. In the same way, any accidental identities can be resolved by user intervention.

Having done this analysis, we are confronted with the problem to create a view of each specific DTD. SIS-Telos offers an elegant and efficient solution. Any object, node or attribute, can be declared as instance of multiple classes. We therefore declare per DTD a trivial metaschema, which classifies all other

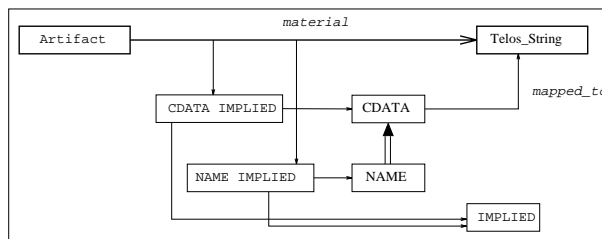


Fig. 6. An example of SGML attributes merging into SIS

constructs generated for this DTD. Shared schema objects are classified under all DTDs sharing it. These metaschemata can be used by any access procedure to filter out the DTD specific or shared schema objects and specific or shared elements of document instances.

In conclusion, we have sketched a conceptual framework to handle different cases of SGML DTD equivalents for merging purposes based on our way of mapping SGML constructs, and which can be elaborated in a cascaded manner by certain interactive and/or automatic processes, depending on the complexity of the case.

7 Implementation of the SGML to SIS Loader

We are currently implementing the necessary parsing and loading tools to handle SGML documents in SIS as described in section 5. The architecture of the SGML to SIS loader is illustrated in figure 7. The main components of the loader are the SGML DTD Analyzer and the SGML Document Parser. Their implementation is based on YACC and LEX. The loading of SGML DTDs and their document instances is performed in two steps:

1. The DTD is analysed by the DTD Analyzer and if it is valid an SIS schema is produced according to the defined SIS metaschema (container and processing part).
2. An SGML document instance of the given DTD is parsed using the Document Parser. If the document is valid then it is loaded in an SIS database according to the generated DTD-specific schema.

Errors are reported at each step of processing. In order to analyse a DTD, the description of the SGML metagrammar in BNF form [22] is used along with the semantic actions associated with the rules of this metagrammar. These actions produce a representation of the DTD in BNF form to generate a DTD-specific parser as well as the necessary semantic actions for the document loading.

8 Conclusions

We have designed a conceptual model to provide dynamic unconstrained schemata for storing, accessing and processing of heterogeneous collections of SGML-encoded documents using DBMS technology. We have shown, that semantic object-oriented data models, such as the SIS data model, can provide more flexible database services for SGML document management than traditional DBMS without the disadvantages of the proposed schema-less approaches [4, 12]. Two features deserve special attention in this context: a) The metamodeling capability of SIS allowing to generate dynamically DTD-specific schemata as instances of a generic metaschema as well as to define an equivalent of the SGML-metagrammar instantiated for each DTD; b) The handling of SIS attributes as objects in their own right allows to express by classification and further attribution complex context relations as in the case of SGML exceptions or DTD-specific views for documents which may be instances of arbitrary DTDs.

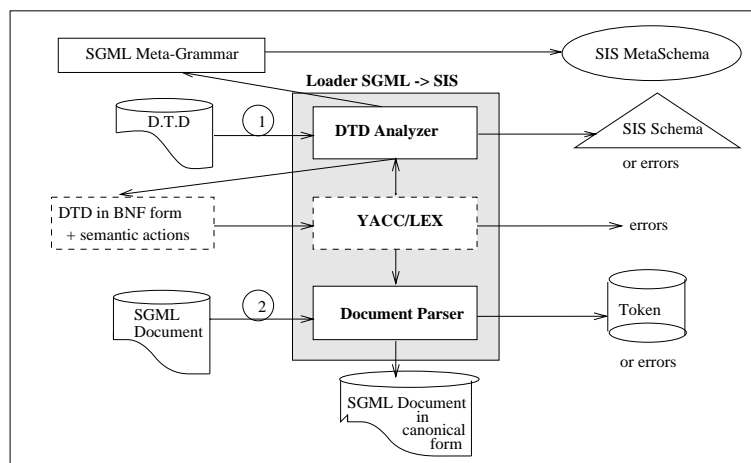


Fig. 7. The architecture of the SGML to SIS Loader

Futhermore the separation of informative structures from control information gives the system an interesting flexibility to provide services of different nature on top of one database. We have then raised some preliminary solutions for the issue of SGML DTD merging. We have shown that the prerequisite for such service is a suitable definition of the information atoms of SGML constructs in a DTD, which can be recognised as identical or distinct as a whole. The contribution of this work is a conceptual framework which makes these constructs explicit and formally accessible at a fine granularity level for optimal processing and sharing of common schema and instance data as expressed by a DTD.

A deeper analysis of the relations emerging from DTD evolution and of viewing mechanisms for heterogeneous documents is planned in the near future and experiences found in distributed database literature are extremely useful in this context. Finally, we must note ongoing studies on the interoperability issues related to the access and querying of heterogeneous SGML documents without a complete knowledge of their structure.

References

1. Description de l'Architecture Générale du Projet GEODOC. Technical report, Grif S.A., 78053 St Quentin en Yvelines Cedex, December 1993.
2. The Extensible Markup Language. Internet Draft, 1997. Available at <http://www.jtauber.com/xml/>.
3. S. Abiteboul. Querying Semi-Structured Data. In Foto Afrati and Phokion Kolaitis, editors, *Database Theory - ICDT'97*, volume LNCS 1186 of *Lecture Notes in Computer Science*, pages 1–18, Delphes, Greece, January 1997. Springer Verlag.
4. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel Query Language for Semi-Structured Data. *Journal of Digital Libraries*, 1(1):68–88, November 1997.

5. A. Analyti, P. Constantopoulos, and N. Spyrtos. On the Definition of Semantic Networks Semantics. Technical Report ICS/TR-187, Institute Of Computer Science - FORTH, February 1997. Available at <http://www.ics.forth.gr/proj/issst/Publications/TechnicalReports.html>.
6. T. Arnold-Moore, M. Fuller, B. Lowe, J. Thom, and R. Wilkinson. The ELF Data Model and SGQL Query Language for Structured Documents. In *Proc. of the Australian Database Conference*, pages 17–26, Adelaide, Australia, January 1995.
7. D. Barnard, L. Burnard, and C. M. Sperberg-McQueen. Lessons Learned From Using sgml in the Text Encoding Initiative. *Computer & Interface*, 18:3–10, 1996.
8. L. Bielawski and J. Boyle. *Electronic Document Management Systems: A User Centered Approach for Creating, Distributing and Managing Online Publications*. Prentice Hall, 1997.
9. G. Blake, M. Consens, P. Kilpelainen, and P. Larson. Text/Relational Database Management Systems: Harmonizing SQL and SGML. In *ADBA '94*, pages 267–280, 1994.
10. K. Böhm, K. Aberer, and E. Neuhold. Administering Structured Documents in Digital Libraries. In *Digital Libraries - Current Issues, DL'94*, Newark, NJ, USA, 1995. LNCS 916, Springer Verlag.
11. M. W. Bright, A. R. Hurson, and S. H. Pakzad. A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, 25(3):50–59, March 1992.
12. P. Buneman, S. Davidson, G. Hillebrand, and D. Sucie. A Query Language and Optimization Techniques for Unstructured Data. In *SIGMOD'96*, pages 505–516, Montreal, Quebec, Canada, June 1996.
13. OCLC Online Computer Library Center. Fred: The SGML Grammar Builder. Available at "<http://www.oclc.org:80/fred/>", 1995.
14. V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *SIGMOD'94*, pages 313–324, Minneapolis, Minnesota, USA, May 1994.
15. V. Christophides, S. Cluet, and G. Moerkotte. Evaluating Queries with Generalized Path Expressions. In *SIGMOD'96*, pages 413–422, Montreal, Quebec, Canada, June 1996.
16. V. Christophides and A. Rizk. Querying Structured Documents with Hypertext Links using OODBMS. In *ECHT'94*, pages 186–197, Edinburgh, United Kingdom, September 1994. ACM.
17. P. Constantopoulos. Cultural Documentation: The CLIO System. Technical Report 115, Institute of Computer Science, FORTH, January 1994.
18. L. Elasy. SGML-DBOO Stockage et Manipulation de Documents Structurés. Master's thesis, Université SORBONE, September 1992.
19. Euroclid. Le Parseur SGML d'Euroclid. Internal document, Euroclid, 12, Avenue des Prés 78180 Montigny le Bretonneux, 1991.
20. P. Francois. Generalized SGML Repositories: Requirements and Modelling. *Computer Standards & Interfaces*, 18:11–24, 1996.
21. P. Fattersack and Q.N. Vuong. Modélisation et Stockage de Documents SGML. Collection de notes internes de la Direction des Études et Recherches 95NO00039, EDF-DER, Service IPN. Département SID. 1 Av. du Général-de-Gaulle, 92141 Clamart Cedex, 1995.
22. C. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.
23. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semi-Structured databases. Stanford Technical Report, 1997.

24. Institute Of Computer Science (FORTH) - Hellas. *SIS - Semantic Index System*, version 2.1 edition, May 1997.
25. ISO. Information Processing-Text and Office Systems- Standard Generalized Markup Language (SGML). ISO 8879, 1986.
26. ISO/IEC. Information Technology-Hypermedia/Time-based Structuring Language (HyTime). ISO/IEC 10744, 1992.
27. P. Kilpeläinen and D. Wood. Exceptions in SGML Document Grammars. Submitted for publication, 1995.
28. R. Light. Getting a handle on Exhibition Catalogues: the Project CHIO DTD. Available at " <http://www.cimi.org/cimi>", Consortium for Interchange of Museum Information, 1995.
29. J. Le Maitre, E. Muriasco, and M. Rolbert. SmlgQL un Langage d'Interrogation de Documents SGML. In *BDA'95*, pages 431-446, Nancy, France, August 1995.
30. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about Information Systems. *ACM Transactions on Information Systems*, 8(4), October 1990.
31. A. Nica and E. A. Rundensteiner. Uniform Structured Document Handling using a Constraint-based Object Approach. In *Digital Libraries: Research and Technology Advances, ADL'95 Forum*, pages 83-101, McLean, Virginia, USA, May 1996. LNCS 1082, Springer-Verlag.
32. D. Raggett. HyperText Markup Language Specification Version 3.0. Internet Draft, March 1995. Available at <http://www.w3.org/hypertext/WWW/Markup/html3/CoverPage.html>.
33. A. Ramfos, N.J. Fiddian, and W.A. Gray. Object-oriented to relational inter-schema meta-translation. In *Workshop on heterogeneous databases*, December 1989.
34. D. Raymond, F. Tompa, and D. Wood. From Data Representation to Data Model: Meta-semantics Issues in the Evolution of SGML. *Computer & Interface*, 18:25-36, 1996.
35. A. Rizk, F. Malézieux, and M. Scholl. Analyse des éléments du Système d'Information: Définition SGML de la Structure des Dossiers de l'Inventaire. Convention de recherche n 295b212 0016008011, Euroclid, 1996.
36. J. F. Roddick. A Survey of Schema Versioning Issues for Database Systems. *Information and Software Technology*, 37(7):383-393, 1995.
37. R. Sacks-Davis, W. Wen, A. Kent, and K. Ramamohanarao. Complex Object Support for a Document Database System. In *Thirteenth Australian Computer Science Conference*, pages 322-333, Victoria, Australia, 1990. Monash University.
38. A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183-236, September 1990.
39. J. Warmer and S. Egmond. The Implementation of the Amsterdam SGML Parser. *Electronic Publishing*, 2(2):65-90, July 1989.
40. D. Wood. Standard Generalized Markup Language: Mathematical and Philosophical Issues. In *Computer Science Today: Recent Trends and Developments*. LNCS 1000, 1995.
41. R. Zicari. A Framework for Schema Updates in an Object-Oriented Database system. In *IEEE Data Engineering Conference*, Kobe, Japan, 1991.
42. J. Zobel, J. A. Thom, and R. Sacks-Davis. Efficiency of Nesting Relational Document Database Systems. In *VLDB'91*, pages 91-102, Barcelona, Catalonia, Spain, September 1991.

This article was processed using the \LaTeX macro package with LLNCS style