ELECTRONIC PUBLISHING, VOL. 6(4), 469–480 (DECEMBER 1993)

The prospects of publishing using advanced database concepts

KARL ABERER, KLEMENS BÖHM, CHRISTOPH HÜSER

GMD-IPSI
Dolivostr. 15
D-64293 Darmstadt, Germany
email: {aberer,kboehm,hueser}@darmstadt.gmd.de

SUMMARY

Publishing is a distributed process which is characterized by the cooperation of different experts. The approach of the Integrated Publication and Information Systems Institute (IPSI) to support electronic publishing is to build an integrated publication environment. The publication of electronic documents demands enhanced support from publishing tools and imposes new challenges on database technology. Taking a hypermedia reference publication as an example, requirements on database technology for the production of electronic publications are discussed. Those can be met by using an object-oriented database management system like VODAK. We present an efficient, flexible and application-independent database application for structured document handling (D-STREAT). Our focus is on dynamic Document Type Definition management.

KEY WORDS SGML Object-oriented database systems Structured document storage Document type definition handling

1 INTRODUCTION

At GMD-IPSI we are developing an integrated publishing environment for the prototypical production of different electronic publications. A sample publication being developed in this framework is an encyclopedic hypermedia reference application for art historians from the Dictionary of Art (DofA), which is a comprehensive reference work to be published (on paper with more than 30.000 pages) in 1996 by Macmillan. [1]

In the past, reference works have been seen as more or less comprehensive and stable views of a subject. Today, there is a shift towards possibly distributed database-aided electronic publications of quality information. The publication content is represented by a knowledge base consisting of an Object Network [1]. Next to the network of Dictionary of Art articles based on the Standard Generalized Markup Language (SGML) [2] it includes a set of interlinked subnets, e.g., index subnet, concept subnet, and a network of domain-specific objects. These objects are, for instance, artists, art styles, institutions, locations, expositions, magazines, works of art, or motifs. Their relationships are expressed

[1] The Application Pilot Dictionary of Art is part of the European RACE (Research and Development in Advanced Communications Technologies in Europe) project 2042 EUROPUBLISHING.

0894–3982/93/040469–12$11.00

Received 15 August 1993
Revised 1 December 1993

by typed links. To create the subnets, extraction of material from different sources will be accomplished using various parsing technologies [3].

The base of material comprises over 11,000 files occupying some 66 megabyte of storage. This represents approximately one quarter of the articles being prepared. The whole reference work has some 16,000 monochrome illustrations. For the electronic version there is considerable leeway for enhancing the interplay of text and factual representations with multimedia content. Examples are color images, graphics, and video/audio annotations.

The resulting requirements for document storage are discussed in Section 2. After the discussion of basic design decisions in Section 3 the next two sections introduce D-STREAT, an application of the object-oriented database management system VODAK being developed at our institute [4,5], to cope with these requirements. Section 4 presents an overview of the modelling of SGML and Section 5 gives a short outlook how to integrate the HyTime Standard [6] into D-STREAT. Finally, Section 6 gives an overview about related work and discusses future work.

2 MEETING THE CHALLENGE OF DOCUMENT STORAGE

To edit and maintain the complex structures mentioned in Section 1, dedicated tools such as the Editor's Workbench [3] are under development. The volume, the fine granularity, and the degree of interconnectivity, i.e., the complexity of the network, makes it impossible to present and view the network as a whole. The editor, therefore, needs to work on individually defined subsets according to the task he is about to fulfill. At the same time, he needs means to browse and update the network freely without being restricted in the degree of granularity.

Such information comprises explicit modelling of relationships between the different kinds of content. Classification of content for the selection and presentation is also mandatory. At any time, the information contained in the database has to be consistent. The information source has to maintain constraints and rules for all kinds of entities involved in the production as well as constraints and rules on meaningful links between entities.

Editing at distributed sites impinges on document storage. The underlying system must facilitate multi-user mode, i.e., ensure consistency maintenance (R1). Concepts to deal with distribution (R2) are required. Functionality to manipulate and navigate hypermedia documents both during the publishing process and in end-user systems is mandatory (R3). Furthermore, immediate meaningful structure-oriented document access is imperative and a descriptive access mechanism is desirable (R4).

Another objective is to capture the semantics of the hypermedia objects within the storage system (R5). The integration of SGML objects is canonical, resulting in, e.g., navigation operations on the document tree. But the constructs of the HyTime standard, associating a standardized semantics to SGML document elements, must be dealt with individually.

Experience with electronic publications [7,8] developed at our institute shows that it is worthwhile not to be restricted to a fixed set of Document Type Definitions (DTDs) (R6). Akpotsui and Quint argue in a similar way [9]: By evaluating users' reactions to (a preliminary version of) their SGML editor they have identified the requirement that DTD alteration by the user must be feasible. Modifying DTDs must not be arduous for the user. Ideally, DTD alteration should not deviate from modification of normal documents. For the management of structured document bases it should be possible – similar to the creation

of tables in a relational database management system (DBMS) – to insert or modify DTDs in the database system without affecting running applications. Documents already stored shall be processable by the system as far as possible after their DTD was modified (R7).

For cooperative editing, version support is advantageous (R8). Versioning enables editors to keep track of their incremental changes and allows to prepare the same content at different levels of detail and abstraction for different products and various groups of users. In another dimension, to handle hypermedia documents the database management system has to be able to deal with multimedia data (R9). Furthermore, it shall be possible to process relations between documents – both of the same DTD as well as of different ones (R10).

3 DESIGN DECISIONS OF A HYPERMEDIA DOCUMENT BASE FOR ELECTRONIC PUBLICATIONS

During the Race Project TELEPUBLISHING [7] and while working on further applications such as the MultiMedia Forum [8] we gathered experience with database support for structured documents by developing the Structured Document Base (SDB) [10]. SDB is a C++ application offering manipulation, navigation, and querying of stored SGML documents. It uses relational DBMSs for persistent document storage. SDB is a realization of the basic SGML standard. It offers extensions conforming to the standard SGML syntax to cope with hypertext links, i.e., machine-supported cross-references, that are inherent in electronic documents. SDB administers SGML-conforming documents in separate pools. Pools correspond to the concept of databases with DBMSs.

Our experiments have shown that the approach taken with SDB has the following drawbacks: SDB serves as a database for SGML documents only, but cannot deal with information not conforming to SGML. Although an extension to HyTime could cover some of the required semantics with respect to, e.g., hypermedia modelling, others cannot be covered within SDB, e.g., the Object Network with its complex relationships, multilingual terminology [11], multimedia content, and text. SDB is realized as an application using DBMS services, but not as a DBMS application. Hence, it has to provide typical DBMS services itself (see R1, R4). Among these services are controlling access to shared data, managing distribution, providing data independence, and versioning [12]. These services aim at the reuse of information for electronic publications. In order to exploit those services within a DBMS application the data model of the DBMS must be expressive enough to capture the semantics envisaged.

The key idea to satisfy the requirements raised in the previous section is to rely on the functionality offered by DBMSs providing expressive modelling primitives. DBMSs allowing data-model extensions are particularly appealing. A general-purpose schema reflects the complex semantics of typed hypermedia objects. Hence, applications are freed from reimplementing these semantics. Therefore, in order to qualify as a basis for a document storage system a DBMS must have certain features to cope with the requirements identified above. First, the requirement that the semantics of hypermedia objects must be captured basically reduces the set of applicable DBMSs to object-oriented ones: Namely, with object-orientation data and the procedures processing it are grouped into autonomous entities, the *objects*.[2] Objects may have *properties* and *methods*. Using methods not only

[2] Principles of object-orientation are generally known, e.g., from object-oriented programming languages. Since terminology, however, is not uniform we briefly establish a "common basis" by reviewing our notion.

data, but also programs are administered by the DBMS. In the database application framework envisaged, these programs realize the semantics of hypermedia objects (R5). On another level, the functionality to, e.g., navigate through the document tree is provided (R3).

The requirements multi-user mode (R1) and distributed mode (R2) can be satisfied in a straightforward way using DBMS standard "built-in features". Transaction management facilitates concurrent access while preserving consistency. Dealing with distribution likewise is standard with DBMSs. The remaining requirements are solved by D-STREAT, which is being introduced in the next two sections. We will focus on R3, R4, R5, structured document handling and DTD management R6, R7, while neglecting other requirements R8, R9, and R10.

4 DYNAMIC SGML DOCUMENT AND DTD STORAGE

The realization of structured document management and the dynamic aspects of DTD handling (R6, R7) strongly depend on the conception of the DBMS used. Within the project HyperStorM ('Hypermedia Document Storage and Modelling') at GMD-IPSI we use the distributed object-oriented database management system VODAK as the basis for D-STREAT. In VODAK, an object's *type* is the set of its properties and methods. *Classes* are a means of abstraction: Objects with the same type may be instances of the same class. Hence, classes are sets of objects. An object is an instance of exactly one class. Since all instances of a class are of the same type, the type definition of instances can be part of the class specification. It is natural to introduce a class corresponding to each SGML element type. Throughout this article, these classes are referred to as *element-type classes*. The instances of such a class are the elements conforming to the element-type definition in a DTD.

To fulfill the dynamic aspects mentioned above the DBMS must allow for the generation of classes at runtime. In this way element-type classes will be automatically created whenever their DTD is inserted into the system. Specific classes for content declaration such as PCDATA or CDATA are created only once for any DTD in the system.

Metaclasses are a VODAK-specific feature. They are classes whose instances are classes themselves. In this context here, it is important that metaclasses have an instance-creation method: The invocation of this method leads to the creation of a new class. Therefore, system shutdown is not necessary to extend the set of element-type classes. The type of the new class as well as the type of its instances is defined in the scope of the metaclass definition. Consequently all element-type classes are of the same type.

Handling of Document Type Definitions. Since it demands a relatively thorough understanding of the principles of object-orientation and the system architecture the users cannot be asked to invoke the methods generating element-type classes themselves. Instead, D-STREAT realizes a more sophisticated conception.

DTDs themselves can be considered as document instances and thus can be rewritten as instances of a particular DTD, the so-called *super-DTD*. In principal, the super-DTD describes the definition of DTDs as defined in the SGML standard. A relevant fragment of the super-DTD is as follows:

```
<ELEMENT elemName=ELEMENT ... contentModel='( ATTRIBUTE *)'>
  <ATTRIBUTE attrName=ELEMNAME attrKeyDecl=NAME attrKeyDef=REQUIRED>
  <ATTRIBUTE attrName=CONTENTMODEL attrKeyDecl=CDATA attrKeyDef=IMPLIED>
  ...
</ELEMENT>

<ELEMENT elemName=ATTRIBUTE ... contentModel=' EMPTY '>
  <ATTRIBUTE attrName=ATTRNAME attrKeyDecl=NAME attrKeyDef=REQUIRED>
  <ATTRIBUTE attrName=ATTRKEYDECL attrKeyDecl=ENUMERATE
    attrRule='(CDATA|ENTITY|ENUMERATE|ENTITIES|ID|IDREF|NAME|NAMES|
    NMTOKEN|NMTOKENS|NO|NOTATION|NUMBER|NUMBERS|NUTOKEN|NUTOKENS)'
    attrKeyDef=UNDEFINED attrDefault='NO'>
  <ATTRIBUTE attrName=ATTRKEYDEF attrKeyDecl=ENUMERATE
    attrRule='(CURRENT|CONREF|FIXED|IMPLIED|REQUIRED|UNDEFINED)'
    attrKeyDef=UNDEFINED attrDefault='UNDEFINED'>
  <ATTRIBUTE attrName=ATTRRULE attrKeyDecl=CDATA
    attrKeyDef=UNDEFINED attrDefault=''>
  <ATTRIBUTE attrName=ATTRDEFAULT attrKeyDecl=CDATA
    attrKeyDef=UNDEFINED attrDefault=''>
</ELEMENT>

<ELEMENT elemName=ENTITY ... contentModel=' PCDATA '>
  <ATTRIBUTE attrName=ENTNAME attrKeyDecl=NAME attrKeyDef=REQUIRED>
  <ATTRIBUTE attrName=ENTTYPE attrKeyDecl=ENUMERATE
    attrRule='(GENERAL|PARAMETER)' attrKeyDef=UNDEFINED
    attrDefault='GENERAL'>
  <ATTRIBUTE attrName=ENTTEXT attrKeyDecl=CDATA attrKeyDef=UNDEFINED
    attrDefault=''>
  ...
</ELEMENT>
```

The fragment shown above illustrates that the super-DTD is conforming to itself, i.e., ELEMENT, ATTRIBUTE, and ENTITY can be described as elements having attributes. Note that with this DTD the attribute definitions belonging to an element-type definition are the content of the element-type definition. This, however, is an arbitrary design decision. According to the SGML standard the corresponding element-type classes are ELEMENT, ATTRIBUTE, and ENTITY. Corresponding to the super-DTD, the instances of ELEMENT have properties containing the values of elemName and contentModel, the instances of ATTRIBUTE have properties for the values of attrName, among others.

Insertion of documents into the system is accomplished with the Amsterdam Parser (ASP) [13] for SGML documents. After verifying the conformance of the document to the relevant DTD an instance of the corresponding element-type class is created for every encountered element in the document.

For the insertion of DTDs the ASP has been extended. The first step when inserting a DTD is checking its conformance to the super-DTD. A result of the parsing process is that original DTDs are transformed to instances of the super-DTD. As an example, consider the following (slightly modified) fragment of the biography-DTD of the DofA.

```
<!ELEMENT HEAD      (NAME, ..., ALTNAME+)>
<!ELEMENT NAME      ♯PCDATA >
<!ATTLIST NAME      TYPE (last|pseudo) "last">
<!ELEMENT ALTNAME   ♯PCDATA >
<!ATTLIST ALTNAME   TYPE (origName|misNomer) "origName">
```

HEAD is the biography head. NAME is the headword or entry of the biography article. It is either a last name or a pseudonym. ALTNAME ("alternative name") could be the original,

civil name, or a misnomer (e.g., Siegfried Bing was often misnamed Samuel). The fragment of a DTD presented above is rewritten as an instance of the super-DTD as follows.

```
<ELEMENT elemName=HEAD ... contentModel='(NAME, ..., ALTNAME+)'>
</ELEMENT>

<ELEMENT elemName=NAME ... contentModel='#PCDATA'>
  <ATTRIBUTE attrName=TYPE attrKeyDecl=ENUMERATE ...
  attrRule='(last|pseudo)' ... attrDefault='last'>
</ELEMENT>

<ELEMENT elemName=ALTNAME ... contentModel='#PCDATA'>
  <ATTRIBUTE attrName=TYPE attrKeyDecl=ENUMERATE ...
  attrRule='(origName|misNomer)' ... attrDefault='origName'>
</ELEMENT>
```

For every element-type definition in the DTD an instance of class ELEMENT is created and the properties are instantiated correspondingly. For every attribute definition an instance of ATTRIBUTE is created and so on. So far, DTD handling does not differ from "normal" document handling.

As a second step, however, a bootstrapping procedure is invoked: For every new instance of ELEMENT an element-type class is created. Now, consider the instances of ATTRIBUTE being part of the content of such an instance of ELEMENT. Every such instance of ATTRIBUTE leads to a property of the instances of the relevant element-type class. From that DTD fragment, element-type classes HEAD, NAME and ALTNAME are created. The instances of NAME have a property named TYPE containing the value "last", the instances of ALTNAME have a property named TYPE containing the value "origName". After these two steps, the system is ready to administer document instances. This approach fulfills R6 in an easy way.

One might wonder how the super-DTD being itself an instance of a DTD is inserted into the system. Since we cannot make out any benefit from being able to modify the super-DTD, the classes such as ELEMENT, ATTRIBUTE and ENTITY are available right from the start. Consequently, modification of these classes is not allowed.

SGML attributes. So far, it has only been mentioned that the system handles the content and attribute values of document elements. In order to illustrate the functionality of D-STREAT it is valuable to disclose some of the internal representation of SGML attributes and content.

SGML attributes are stored as a list of key/value pairs. The keys of type STRING contain the attribute names. Since in the original SGML document attribute values are just a sequence of characters values are also of type string. The list is a VODAK built-in datatype.

With SGML it is basically the representation of the documents' structure that can be standardized. The interpretation of some standard types such as ID, IDREF is canonical, the interpretation of user-defined attributes cannot be described. Hence, this generic modelling of the SGML attributes is appropriate. As opposed to this, for modelling HyTime a semantically enriched model is appropriate (cf. Section 5).

SGML content. Representing the content of SGML elements is rather straightforward. Similar to the instances of the element-type classes the content parts are instances of the specific classes for the content declarations PCDATA or CDATA.

Incremental updates. If we would limit ourselves to leave the documents unmodified after inserting them into the document base, the storage of content models and attribute models of the element declaration would not be necessary. However, our objective is to enable modifications of document components without repeating the process of parsing the entire document and re-inserting it into the document base. In order to check an element's conformance to the DTD, the relevant content model and attribute model must be available. In our approach, the content models and attribute models are kept as properties of the element-type classes so that the DTDs are available in a quasi-compiled form.

```
<HEAD ...>
  <NAME TYPE=last ...>
  Bing
  </NAME>
  <ALTNAME TYPE=misNomer ...>
  Samuel Bing
  </ALTNAME>
</HEAD>
```

Before discussing the functionality of the system envisaged (R3,R4), we give an example of how SGML documents and DTDs are stored. Consider the following fragment of a document instance conforming to the DTD fragment presented above.

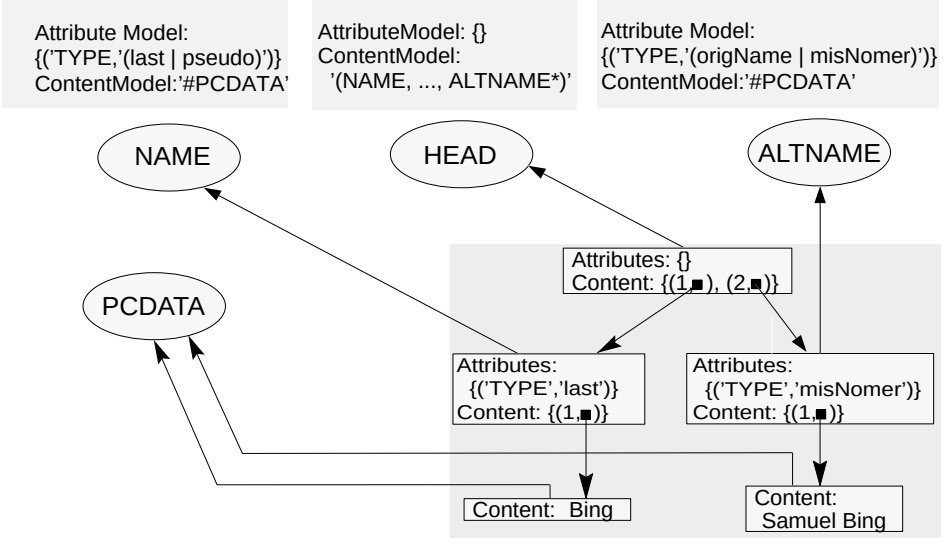

Figure 1. SGML document storage

The instance is stored as depicted in Figure 1. Classes are represented by ellipses. Instances (that are not classes) are shown by rectangles. An instance is connected to its class with a plain-line arrow. There are classes HEAD, NAME and ALTNAME. The instances of class HEAD are elements of type HEAD, the instances of class NAME elements of type NAME etc. The properties of an object are shown within rectangles (in case of a plain object) or right above the ellipse (in case of a class). The grey box at the bottom right corresponds to the document fragment shown above.

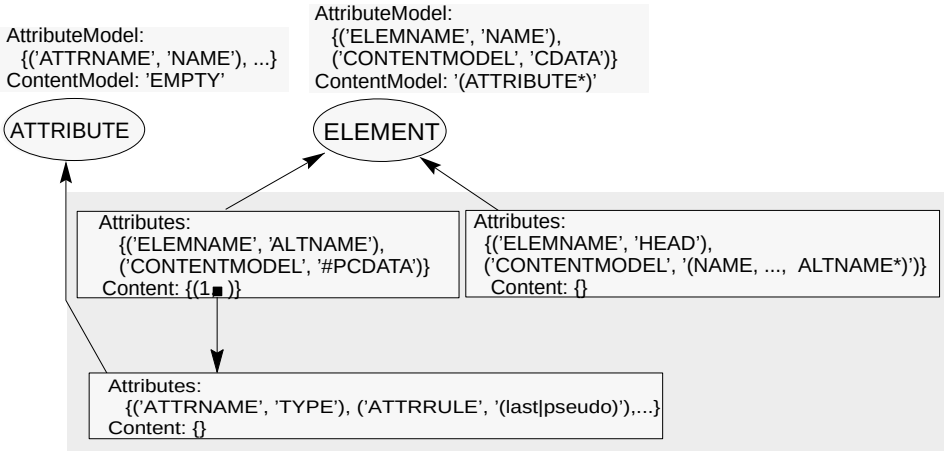

Figure 2. SGML DTD storage

The storage of that DTD fragment with element-types HEAD and ALTNAME follows the same principle. It is displayed in Figure 2. The grey box is a fragment of the DTD fragment from above.

The functionality of the document base is illustrated by means of an example: Suppose that in the sample document above we want to insert another misnomer ('Sam Bing') for Siegfried Bing. The sequence of operations is basically as follows:

```
BEGIN_TRANSACTION;
   new_altname := ALTNAME -> createElem();
   flag := new_altname -> setAttribute ('TYPE', 'misNomer');
   flag := head -> subElemInsert (new_altname);
   new_pcdata := PCDATA -> createElem();
   flag := new_pcdata -> setTextualContent ('Sam Bing');
   flag := new_altname -> subElemInsert (new_pcdata);
END_TRANSACTION;
```

In the notation used above the target object of a method is left to the arrow, the method with its parameter on the right. ALTNAME, new_altname, new_pcdata, and head are variables of type OID, the unique object identifier in VODAK, flag is of type BOOL. ALTNAME is the OID of the element-type class with the same name (cf. Figure 1), head the OID of the instance of HEAD that already exists (i.e., the instance of HEAD in Figure 1). The method createElem creates an instance of the target object, which is a class. The first parameter of setAttribute is the attribute name, the second parameter the value. subElemInsert inserts the parameter, an object, into the content of the target object. setTextualContent, a method that only exists for instances, sets PCDATA content. By bracing the operations with **BEGIN_TRANSACTION** and **END_TRANSACTION** this sequence of operations becomes an atomic unit: The DBMS guarantees that either all operations or none of them are executed and that their execution is not interrupted by other operations. This functionality facilitates multi-user mode that cannot easily be achieved with a file-based system.

Queries. Data manipulation (R3) and retrieval (R4) can be specified in a declarative way using query languages. The path orientation of the VODAK Query Language (VQL) fits

the conception of SGML and the location model concept of DSSSL [14]. The following sample query returns all instances of element-type ALTNAME being misnomers:

```
ACCESS a
FROM a IN ALTNAME
WHERE ((a -> getAttributeValue('TYPE')) == 'misNomer')
```

This query illustrates that method calls can be part of query expressions with the underlying object-oriented DBMS: The parameter of method getAttributeValue is an attribute name, the return value the corresponding value. The syntax is analogous to SQL and thus not explained. The following, more complex query expression is an example of how path expressions can be used in the SGML context:

```
ACCESS h
FROM h IN HEAD
WHERE ((((h -> getContentElement(1)) -> getContentElement(1)) ->
    getTextualContent()) == 'Bing')
```

The method getContentElement with parameter 1 returns the first content element of the target object, which is an SGML element. getTextualContent is the counterpart of setTextual-Content. Note that the instance of HEAD shown in Figure 1 belongs to the values the query returns. This example also illustrates how knowledge of the document type can be used to formulate queries: We know that the NAME element is first in the content of HEAD elements.

Modification of DTDs. Since DTDs are considered as documents, they can be modified as any other document (R7). Suppose we want to alter the content model of HEAD from (NAME, ..., ALTNAME+) to (NAME, ..., ALTNAME*) can be done by invoking the following operation:

```
flag := elem -> setAttribute ('CONTENTMODEL', '(NAME, ..., ALTNAME*)');
```

elem is the OID of the object shown in Figure 2 being the instance of ELEMENT on the right hand side.

The alteration must be propagated to the relevant element-type class. The sample modification is not problematic because it is an extension operation. In general, however, modifying an element-type definition or attribute definition may have the effect that the instances are not conforming to the DTD any more. For example, changing the content model from (ALTNAME*) to (ALTNAME+) causes a problem if there are sections whose content is empty. For the moment it is the system administrator's responsibility to ensure that only extension operations occur to DTDs that already have instances. Shifting this responsibility to the system, however, is subject to further work (cf. [9]).

5 HYTIME

The next step to integrate more semantics of hypermedia objects within the storage system (R5), is to extend D-STREAT towards HyTime support. In a nutshell, the HyTime standard is a list of element-type definitions. Using HyTime terminology, these templates are *element-type forms* or, more generally, *architectural forms*. [3] Element-type definitions in application

[3] There are two kinds of architectural forms: *Element-type forms* and *attribute list forms*. In this paper we limit ourselves to element-type forms.

DTDs may have as their core an element-type form together with further attributes. These element-type definitions are specializations of the element-type form. The decisive point is that the semantics of the content and attributes of an element-type form is standardized independently of any DTD. For instance, consider the element-type form clink which we took from [6]:

```
<!element ilink  -- Contextual link --
                 - O        (%HyBrid;)*>
<!attlist ilink  HyTime     NAME     clink
                 id         ID       #IMPLIED
                 linkend    -- Link end --
                            ...
                            IDREF    #REQUIRED>
```

clink is the HyTime element name for 'contextual link'. A contextual link is a reference together with content. Footnotes are a typical example. The attribute linkend contains the ID of the document component that is referenced. '(%HyBrid;)*' basically implies that the content of clink-instances is in essence arbitrary. To continue the example, it may make sense to mark the components of a document (esp. of a document that is written by several authors) with annotations. A sample element-type definition using clink might be as follows:

```
<!ELEMENT annotation    #PCDATA >
<!ATTLIST annotation    HyTime    NAME    clink
                        id        ID      #IMPLIED
                        linkend   IDREF
                        author    NAME>
```

While the attributes HyTime and linkends are inherited from the HyTime element-type form, the attribute author is part of that particular clink-specialization only. Likewise, the content of type CDATA is a specialization of (%HyBrid;)*. An instance of annotation would look exactly like an instance of an SGML-element-type definition. The fundamental difference compared to SGML is that the semantics of architectural forms is standardized. A trivial example of an operation performing application-independent clink-processing would return the document component that is referenced by a given contextual link.

The internal representation of HyTime is as follows: For every HyTime element-type form there is a metaclass in the document base. We refer to these metaclasses as *HyTime metaclasses*. To each SGML element-type being a specialization of a HyTime architectural form corresponds an element-type class and an instance of the relevant HyTime metaclass. These instances are called *HyTime element-type classes*. Like element-type classes (cf. Section 4) HyTime element-type classes are created at runtime. Every SGML document element being an instance of a HyTime element-type form has two objects in the document base representing it: An instance of the element-type class and an instance of the HyTime element-type class. The instances of the HyTime-element-type classes have methods capturing the HyTime processing semantics. Moreover, to support more semantics (R5) the integration of multimedia data into VODAK [15,16] can be exploited.

6 CONCLUSIONS

The paper introduced D-STREAT, a prototypical VODAK application framework for the storage of SGML/HyTime documents. D-STREAT handles instances of arbitrary document type definitions. Instances of element-type definitions are created dynamically. Their structural power is well appreciated, for example, by retrieval operations. On the SGML-side the system provides SGML-specific functionality, e.g., navigating on document trees. Due to lack of space the HyTime aspect has only been summarized in this paper. For the same reason we just mention that inter-document processing is also possible with our approach (cf. R10).

In [17], the object-oriented approach has been identified as the most suitable approach to hypertext storage. ODA (Office Document Architecture) is another standard for document description similar to SGML. Brown et al. [18] have realized an ODA-document-storage system. It consists of a so-called Object Manager and a so-called ODA Manager. With the Object Manager C++-objects being ODA-document components are made persistent. The ODA Manager provides the ODA-specific semantics to those objects. The objective is to provide an "intelligent" interface to ODA documents such as D-STREAT provides for SGML/HyTime documents.

In [19] the conception of a 'client-server-architecture' in the document-handling context is as follows: The client is an application process, e.g., a particular user interface. The server performs application-independent document processing (e.g., navigation on the document structure). With that terminology, our approach can also be classified as a client-server architecture. To extend the described functionality in [19], the need for databases (or at least research results from this area) is acknowledged. In this sense our work shows the prospects of publishing using advanced database concepts.

We claim that a new facet of our approach is that arbitrary DTD instances can be handled. This has been achieved by exploiting the 'metaclass'-features of the underlying object-oriented DBMS VODAK. As mentioned in the requirements section and looking at further experiences made in the DofA project [3] more complex document semantics not yet covered in HyTime have to be administered by D-STREAT. Those semantics can easily be integrated by object-oriented database systems such as VODAK. Future work is to support not only interdocument relations between SGML/HyTime documents but to allow interconnections between D-STREAT and different document data models.

7 ACKNOWLEDGEMENTS

We like to thank Wiebke Möhr and Anja Haake for their comments on earlier versions of the paper.

REFERENCES

1. C. Hüser et al., 'AP1: Requirements, Publishing Process and Environment', Technical Report R2042/GMD/IPS/DS/R/008/b1, EUROPUBLISHING Project, RACE-Programme, (1993).
2. *International Standard ISO/IEC 8879: Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*, ed., International Organization for Standardization, Geneva/New York, first edition, 1986.
3. L. Rostek, W. Möhr, and D.H. Fischer, 'Weaving a web: The structure and creation of an object network representing an electronic reference work', *Electronic Publishing – Origination, Dissemination, and Design*, **6**(4), (December 1993).
4. W. Klas et al., 'VML - The VODAK Model Language Version 3.1', Technical report, GMD-IPSI, Darmstadt, (July 1993).
5. W. Klas, K. Aberer, and E.J. Neuhold, 'Object-Oriented Modelling for Hypermedia Systems Using the VODAK Modelling Language (VML)', in *Object-Oriented Database Management Systems, NATO ASI Series*, Springer, Heidelberg/Berlin/New York, (August 1993).
6. *International Standard ISO/IEC 10744: Information Technology – Hypermedia/Time-based Structuring Language (HyTime)*, ed., International Organization for Standardization, Geneva/New York, first edition, 1992.
7. C. Hüser and A. Weber, 'The Individualized Electronic Newspaper: An Application Challenging Hypertext Technology', in *Hypertext und Hypermedien 1992: Konzepte und Anwendungen auf dem Weg in die Praxis*, eds., R. Cordes and N. Streitz, 62–74, Springer, Heidelberg, (1992).
8. K. Süllow et al., 'MultiMedia Forum – an interactive online journal', *Electronic Publishing – Origination, Dissemination, and Design*, **6**(4), (December 1993).
9. E. Akpotsui and V. Quint, 'Type Transformation in Structured Editing Systems', in *Proceedings of Electronic Publishing, 1992 (EP92)*, eds., C. Vanoirbeek and G. Coray, 27–41, Cambridge University Press, Cambridge, UK, (1992).
10. C. Hüser, 'Report on a prototypical interface for structured documents and its application to the IEN scenario', Technical Report 75/GMD/IPS/DS/L/047/b0, TELEPUBLISHING Project, RACE-Programme, (August 1991).
11. D.H. Fischer, 'Consistency Rules and Triggers for Multilingual Terminology', in *Proceedings of the Third International Congress on Terminology and Knowledge Engineering*, 333–342, Indeks Verlag, Frankfurt am Main, (1993).
12. A. Haake, 'CoVer: A Contextual Version Server for Hypertext Applications', in *Proceedings of the Fourth ACM Conference on Hypertext (Hypertext'93)*, 43–52, ACM Press, New York, (December 1993).
13. J.Warmer and S.van Egmond, 'The implementation of the Amsterdam SGML Parser', Technical report, Faculteit Wiskunde en Informatica, Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam, (1987).
14. *International Standard ISO/IEC DIS 10179: Information Technology – Text and Office Systems – Document Style Semantics and Specification Language (DSSSL)*, ed., International Organization for Standardization, Geneva/New York, 1991.
15. T.C. Rakow et al., 'Using Object-Oriented Database Systems for Multimedia Applications', in *it + ti – Informationstechnik und Technische Informatik, Themenheft "Multimedia/Hypermedia", Teil 2*, 4–17, Oldenbourg, Munich, (June 1993).
16. K. Aberer and W. Klas, 'The Impact of Multimedia Data on Database Management Systems', Technical report, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Sankt Augustin, (1993). Arbeitspapiere der GMD, No. 752.
17. D.B. Lange, 'Object-Oriented Hypermodeling of Hypertext Supported Information Systems', in *Proceedings of the 26th Hawaii International Conference on System Sciences*, (1993).
18. A.L. Brown, T. Wakayama, and H.A. Blair, 'A Reconstruction of Context-Dependent Document Processing in SGML', in *Proceedings of Electronic Publishing, 1992 (EP92)*, eds., C. Vanoirbeek and G. Coray, 1–25, Cambridge University Press, Cambridge, UK, (1992).
19. R. Furuta, P.D. Stotts, and G.D. Drew, 'Experiences with a Client-Server-Based Architecture for a Distributed Structured Hypertext System', in *Proceedings of Electronic Publishing, 1992 (EP92)*, eds., C. Vanoirbeek and G. Coray, 113–141, Cambridge University Press, Cambridge, UK, (1992).