Administering Structured Documents in Digital Libraries*

Klemens Böhm, Karl Aberer, and Erich Neuhold
GMD-IPSI
Dolivostraße 15, 64293 Darmstadt, Germany
E-mail: {kboehm, aberer, neuhold}@darmstadt.gmd.de

Abstract

In this chapter we argue that hyperdocuments administered by digital libraries have to be structured according to standardized storage and exchange formats in order to allow for the manipulation functionality required in digital library construction, maintenance and use. We demonstrate how SGML and its extension, HyTime, can play this structuring role, and how multimedia documents structured accordingly can be stored, changed and maintained in the object-oriented database system VODAK. Using the dynamic semantic extension facilities of VODAK it is illustrated how the document structuring dynamics offered by SGML and HyTime can be accommodated in the database. In addition we discuss how this facility can be combined with other system components to provide a relevant portion of the functionality required for digital libraries.

1 Introduction

The exponential growth in the amount of published information has led to a crisis in many of today's libraries. On the one hand the choice of what to keep, even in a limited field, is becoming harder and harder, on the other hand the selection of relevant information for a specific library user also becomes more difficult. The old idea that a library provides its readers with the right information in the right amount for an acceptable price at the right time is about to fail. In a nutshell, a solution to these problems can be seen in transforming the contents of a library to digitalized electronic form and thus gaining advantages in producing, selecting, storing and offering information. However, next to the growth in the amount of information two other phenomena have occurred. The

*appeared in: Advances in Digital Libraries, Lecture Notes in Computer Science 916, Springer Verlag, 1995.

information to be handled by libraries is increasingly becoming multimedia. In addition to texts, also images, videos, audios, even animation- and simulation programs, e.g. computer games, have to be kept. Second, this multimedia information is increasingly becoming interconnected, resulting in hypermedia documents. Inter- and intra-document links have to be created between distant storage locations and maintained sometimes over long periods of time. One of the frequently cited examples of this information interlink is the World Wide Web, the world-spanning "hyperdocument". However, the idea of essentially offering the worlds information as a single "document" containing innumerable pieces of information interconnected by an innumerable amount of links does not solve the crisis mentioned above. It rather aggravates the problem. To offer the user the right information in the right amount at the time of need is becoming more difficult as the World Wide Web is continuing to grow in an uncontrolled way. Anybody is now free to produce and offer arbitrary "information" to the whole world. The human being, however, who needs information for his work, his decision making or entertainment planning is actually the loser as many people have observed. Browsing, i.e. not searching (!), the Internet we have found the following characteristic description of the situation.

> From: rocky@cadence.com (Rochelle Grober)
>
> "The Internet is already an information superhighway, except that you have to be a full-fledged computer nerd to navigate it. I have been there. It's like driving a car through a blizzard without windshield wipers or lights, and all of the road signs are written upside down and backwards. And if you stop and ask someone for help, they stutter in Albanian."
>
> Columnist Mike Royko

In this chapter we cannot possibly discuss all these aspects of the electronic information age, and the new roles all players in the information chain have to assume. We will concentrate on the document infrastructure that we are developing as a basis for the creation, interchange, storage and offering of digital multimedia interlinked documents. The multitude of other services in the "digital information world" will basically be ignored for now. But, as we believe, they can be added easily to the framework we are discussing here. Two large experiments at our research laboratory are currently used to evaluate this assumption. An electronic house-magazine, the MultiMedia Forum [27][1], evaluates concepts and ideas of distributed electronic authoring, cooperative editing, and releasing information in digital newspapers or digital magazines. A project centered around the Dictionary of Art [24], a digitized encyclopaedical information store, is focusing on the interoperation of authors, editors, publishers and eventually of information brokers, a possible new role of libraries and librarians, together

[1] A public version of the MultiMedia Forum is available on WWW under http://www.darmstadt.gmd.de.

with a variety of users. The individualization of the information offered to the end users is the principle goal - and added value - of this information store.

To handle documents electronically a digitized format for all documents is necessary. They may already be produced in digitized form - an approach that does hold true for texts, images, audio, video or any other representation medium of information. On the other hand, they may also be produced in a "traditional" way and then have to be converted into a digital format. In both cases individuals using different hardware, software and other machinery platforms are involved in handling large or even huge amounts of information. In order to create, exchange, control and use this information it has to be structured into manageable pieces using certain conventions or at least conversion mechanisms that are agreed upon between the many different representation and interchange formats available. When not considering a document to be 'atomic' but rather structured, more flexible access mechanisms to documents are possible [6, 11, 8]. Next to documents' classification by subject, a traditional task of publishers and libraries, structured information can be used to locate and retrieve documents more effectively. For example, it is possible to retrieve only the abstract of "full text" documents, to retrieve all chapter headings, or to use full text search mechanisms to retrieve those chapters or sections only containing specific information. In an interlinked hyperdocument such "scope" restrictions will be needed to limit the size of the document pieces selected, transported, stored and displayed in response to an information retrieval query. Limiting the size of the retrieved documents through navigation along intra- and inter-hyperdocument links leads to the "driving" phenomena encountered in the WWW or observed as "getting lost in space" [12] in hypertext systems.

Problems related to documents' internal structure have become more acute due to the increasing supply of information. Hence, the demand for concepts and technologies has also risen. SGML ('Standard Generalized Markup Language') [ISO86, Her90] is within the center of interest. With SGML the logical structure of documents of arbitrary types[2] can be described. In a nutshell, this is accomplished by identifying logical document components such as sections, subsections, definitions, listings, figures, etc. within the document. SGML documents need not be text documents. Rather they may be composed of basic components of arbitrary types, e.g. pixel graphics or audio. - However, in SGML document-type definitions the semantics of document components are not recorded. As a consequence, rules for handling the different datatypes are not part of SGML.

In more detail, on the one hand, there is the processing semantics, e.g. how to handle a video, an audio, or a simulation program. Besides that, there is the 'real world' content semantics of document components, i.e. the role of document components such as title, abstract or conclusions. Other examples are

[2] At a rudimentary level of analysis, examples of document types are letter, technical manual, newspaper, biography, novel.

the name of a letter's sender or in a book the information about the publisher or author, together with the knowledge how to handle this information. For example, one may store the sender's address in an address register, or one will pay the publisher royalties when accessing the electronic version of the book.

Processing semantics have now been included into SGML via the HyTime standard. It contains a number of templates to define canonical hypermedia elements such as (time-)schedules and hyperlinks. These semantics, but also the content semantics, must be reflected before full manipulation functionality can be achieved. Capturing the content semantics is closely related to the field of document content description by authors, editors, publishers, librarians, etc. or document content analysis by computer linguists, indexers, document relevance analyzers, etc. Standardization of content semantics in the form of knowledge nets is currently out of reach by much. Content semantics will not be further considered in this chapter.

Even though approaches for the identification of logical document components are subject to these restrictions, we make a strong point for the use of such formats when administering documents in digital libraries: The dividing line between intra-document relationships and inter-document relationships is not clean-cut. As an example, consider a document of type 'conference proceedings' consisting of several articles. On the one hand, these articles are parts of the document of type 'conference proceedings'. On the other hand, they can be seen as documents in their own right. The impact of these different views is that the same portion of the digital library may be referenced in different ways.

Another consequence of structuring documents in digital libraries is that search and delivery size of information can be limited.

The equation of 'libraries' with 'archives', that can be encountered in the literature occasionally, leaves aside certain subtleties. Archives are static, i.e. documents that have been archived are normally not altered. In order to modify them in general they need be dearchived. To support this rather static view, less sophisticated architectures tend to be sufficient. With libraries, however, reorganizing the document stock, modifying individual documents, and annotating them by the original author, by readers, or by other authors must be possible. Database management systems (DBMSs) should be applied to meet the requirements imposed by this more sophisticated functionality. Databases allow for concurrent use and update, for user protection, for recovery and persistence of the information stored. If database technology was not used, these properties would have to be provided by the individual application systems accessing the digital library.

The approach taken in this paper is to describe first our way on how to store SGML documents in an object-oriented database. An extension towards HyTime is sketched afterwards as it has not yet been implemented completely. Subsequently we illustrate, by a number of sample applications, how the hyperdocument base may be combined with other system components to cover the integral handling of documents in various applications in a more complete way.

Our application framework stores documents using the object-oriented database management system (OODBMS) VODAK [20, 18]. As described above, the documents that are particularly well-suited for storage in object-oriented databases are highly structured and are repeatedly edited or annotated by several persons. Typical examples are newspapers changing dynamically ("partial news-update-strategy") or encyclopediae with many authors and editors and individualized or annotated versions for different user groups.

Conventional file-oriented systems without databases [25] lead to many difficulties in handling this kind of documents. For example, by leaving document-file objects intact multi-user mode is not supported. The granularity is on the document level. Concurrently authoring different parts of the same document is cumbersome at least. Besides that, if versioning is to be supported then each task has to have a copy of the entire document even if only a small fragment of the document will be modified. On the other hand, fragmentation of documents in database applications, as practised so far [14, 29, 13], also has a number of shortcomings. For example the question how documents are fragmented in an optimal way cannot be generally answered. Either a very generic document structure has to be chosen, or only a few predefined specific document types can be supported. From our point of view, this is the main distinction between the notions to be presented in this article and other approaches to store documents within databases. While others limit themselves to design database schemata for just one or a few document types our system can handle documents of arbitrary types. Furthermore, by designing database schemata for individual document types, it is implicitly assumed that document types do not change over long periods of time. If the system is based on a very generic document type, this view might actually be sufficient, but this is not sufficient for a more refined approach to document handling [5].

In our approach logical document-component types of SGML correspond to classes in the database. To be capable of handling arbitrary document types and of incorporating document-type changes into the system these classes are created dynamically and may in principle be modified at runtime. The underlying OODBMS VODAK provides the relevant system support. SGML document-type definitions (DTDs) introduce the logical components of a document of a particular type. Hence, with this approach the decision how the documents are fragmented in the document base is shifted to the designer of the DTD. In this article we will describe the kernel structure of our application framework and the functionality it offers to the applications illustrated. Another facet, dynamic DTD handling, has already been discussed in [2].

The remainder of this article is organized as follows: In the following section SGML concepts are briefly reviewed in brevity. Section 3 contains a short overview of the VODAK Modeling Language. Both Section 2 and Section 3 form the basis to describe our database application framework. This description is given in Section 4. Section 5 deals with the integration of HyTime semantics. In Section 6 the additional functionality which may be obtained by coupling

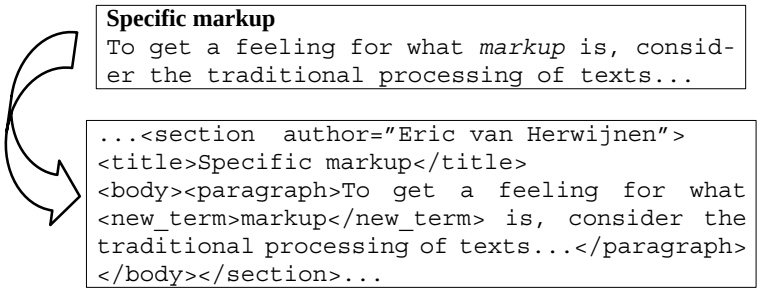

Figure 1: SGML Example: Markup

the framework with other systems components is documented. Conclusions are contained in Section 7.

2 Review of Basic SGML Concepts

SGML (Standard Generalized Markup Language) has received considerable attention, because it provides the means to tackle some of the problems related to document handling. In the SGML context, the distinction between a document's *logical structure* and its *layout structure* is fundamental: Italicizing or boldsetting a word is on the layout side. Identifying the reason why a word should be italicized, e.g. because it is introduced right there, is on the logical level. This identification can be accomplished by *marking up* the document. In the document fragment in the upper right of Figure 1 `<title>` and `<body>` are examples of *start tags*, `</title>` and `</body>` are the corresponding *end tags*. With SGML a document's logical structure can be predefined. A document consists of document elements, e.g. `title, body, new_term`. The relationship between them is specified by a set of production rules called *document type definition (DTD)*. Figure 3 contains a fragment of the DTD of the document instance from Figure 1: A section consists of a title followed by a body. A paragraph's content is a list whose components are either `CDATA` elements or new terms. Lists are defined using '*', alternatives by using '|'. In a nutshell, `CDATA` is a "plain data type" comparable to STRING. Sections have an attribute author of type `CDATA`. - From another perspective, DTDs are machine-independent document-interchange formats.

Using SGML inter alia has the following advantages.

- Authors can concentrate on the document content rather than its layout.
- We call the fact that the same layout is used for the same logical document components *document consistency*. It is fostered by using markup instead of "private" layout conventions. If big documents are composed

Misuse of SGML concepts: Markup does not capture the logical structure

```
<bold>Specific markup</bold>

To get a feeling for what <italic>markup</italic>

is, consider the traditional processing of texts...
```

Figure 2: SGML Example: Misuse of Markup

```
<!ELEMENT document  section*>
<!ELEMENT section   (title, body)>
<!ATTLIST section   author             CDATA>
<!ELEMENT title                  CDATA>
<!ELEMENT body                   paragraph*>
<!ELEMENT paragraph (CDATA|new_term)*>
<!ELEMENT new_term  CDATA>
```

Figure 3: SGML Example: Fragment of an SGML Document Type Definition

without using markup it can easily occur that one author, say, underlines new terms and another one italicizes them. Document consistency is an issue both within documents as well as among several documents. It is a contribution to corporate identity.

- A marked-up document bears more information than one without markup. With markup authoring is eased especially if there are several authors or other persons involved in the process, e.g. reviewers.
- Queries making use of the document structure can be formulated, e.g. "Select all new terms." or "Select all sections whose title contains the word 'markup'.". SGML is most useful to identify logical document components whose function is not evident even to a human reader. In a big reference work there may be two dozens of reasons why, say, a word might be italicized.
- Attributes for document elements can be introduced (see Figure 1 and Figure 3 for attribute `author`). This further eases (multi-)authoring and querying.

SGML Terminology. In SGML a document has a tree structure: the nodes, i.e. the logical document components, are called *elements*. The subelements of an element are its *content*. The leaves contain the *data content*. In Figure 1 there are elements `section`, `title`, `body` etc. There is a differentiation between *generic* and *specific document descriptions*. For instance, the specification that a section contains a list of paragraphs is generic. On the other

hand, stating that a section consists of a particular paragraph p1, followed by paragraph p2, is specific. An *element-type definition* is the generic description of an element, i.e. the set of rules specifying its content and attributes. The element-type name is also called *generic identifier*. The generic description of the content of elements of a particular type is its *content model*, the one of the attributes the *attribute model*. In SGML there exist six *constructors* to construct a content model from other element types, three *connectors* and three *occurrence indicators*. For instance, the *sequence connector* (,) introduces an order of the content-element types, '*' is the *optional and repeatable occurrence indicator*.

`CDATA` and `#PCDATA` being examples of *terminal element types* contain textual data. - Aside from SGML there is the standard ODA [16] and various proprietary formats to describe documents' logical structure. The concepts that are discussed in the sequel can quite easily be applied to other formats.

3 Key Concepts of the VODAK Modeling Language (VML)

Applying OODBMSs has turned out to be advantageous with regard to various application domains. Because there exist conceptual and terminological differences between different OODBMSs and OOPLs the terminology of the VODAK Modeling Language (VML) is reviewed in brevity. With object-oriented models the data and the procedures that process them tend to be grouped in autonomous entities, the *objects*. We call the constituents of an object *properties* and *methods*. Properties are the variable-like containers for the data, methods the procedures capturing objects' semantics. In our terminology, the object's *type* is its property- and method definitions. As usual, objects' unique identifiers are given out and administered by the system. In the VML conception, *classes* are sets of objects of the same type. In VML, it is possible that instances of different classes are of the same type. The separation of the structural and the extensional aspect is called *dual model*. With main-stream OODBMSs this differentiation is not made. In VML classes are first-class objects. With the OODBMS VODAK both data and operations on it are administered by the system. The advantage is that the application semantics is within the database system. A sample code fragment is given in Figure 4. The INTERFACE-part is the public part, the IMPLEMENTATION-part the private one. The metaclass of the class is preceded by the keyword METACLASS. METACLASS is a metaclass provided by the system that does not furnish a particular semantics for its instances and metainstances.

Metaclasses are a special feature of VODAK. A metaclass is a class whose instances are themselves classes. Symmetrically, a *metainstance* is an instance of a metaclass's instance. As a rule, a class definition contains the definition

```
CLASS PARAGRAPH METACLASS METACLASS
    INSTTYPE
        PARA_INSTTYPE;
END;

OBJECTTYPE PARA_INSTTYPE
    INTERFACE
        METHODS
            retrieveTextualContent(): STRING;
            ...
    IMPLEMENTATION
        PROPERTIES
            content: STRING;
        METHODS
            retrieveTextualContent(): STRING;
            ...    //method implementation omitted
END;
```

Figure 4: VML Code Fragment

```
CLASS CATSPEC METACLASS METACLASS
    //This metaclass is metaclass of both gener-
    //alization and specialization classes. Namely,
    //in multi-level inheritance a class can be both
    //generalization and specialization class.
    INSTTYPE CATSPEC_INSTTYPE
    INSTINSTTYPE CATSPEC_INSTINSTTYPE
END;
```

Figure 5: VML Code Fragment of a Metaclass

of its instances' types. In VML terminology, this type is the *insttype* of that class. In addition, it is possible that an object has properties or methods the other instances of its class do not have. (This is a relaxation of the principle that all instances of a class are of the same type.) Those individual properties and methods are part of an object's *owntype*. Furthermore, the properties and methods in a metaclass's *instinsttype* are the ones of its metainstances. Hence, an object has the properties and methods defined in its owntype, in its class's insttype and in its metaclass's instinsttype.

Inheritance. In VML there is a distinction between three kinds of *inheritance*. Two of them are mentioned here: *type inheritance* and *inheritance via metaclasses*. It is possible to factor out a portion of an object-type definition and to reuse it in other object-type definitions. This mechanism is called type inheritance or *subtyping*. On the other hand, the definition of a metaclass contains properties and methods of their instances and metainstances. The phenomenon that an object or an application class has properties and methods that are neither part of the object definition nor the class definition but instead part of a metaclass definition is referred to as inheritance via metaclasses.

```
OBJECTTYPE CATSPEC_INSTTYPE
    INTERFACE
    METHODS
        defHasSpecCls(specCls: OID): BOOL;
            //This method is applied to a generalization class to specify that the class
            //'specCls' is one of its specialization classes.
        defHasGenCls(genCls: OID): BOOL;
            //Symmetrically, the target is the specialization class, and the parameter
            //('genCls') is the generalization class.
        ...
    IMPLEMENTATION
    PROPERTIES
        hasCatSpecClasses: {OID};
            //This property contains the specialization classes.
            //The property values are the OID of the specialization classes.
        GeneralizationClass: OID;        //symmetrically...
        ...
END;
```

Figure 6: VML Code Fragment of a Metaclass's Insttype

Semantic Relationships. A reason why metaclasses are in use is to model semantic relationships between classes. Examples of semantic relationships are *aggregation ("partOf")* or *specialization*. For example, a section may be an aggregation of subsections, a subsection an aggregation of chapters, and a chapter an aggregation of paragraphs. Some OODBMSs offer hardcoded mechanisms to describe relations between classes (cf. IS-A and IS-PART-OF relationships [7]). However, semantic relations have a variety of facets. Furthermore, some facets, which are called dimensions in [15], impinge on the interface. E.g. within an aggregation the order of the components may be relevant (as with the chapters of a book) or not (as with the ingredients of a fruit salad). With hardcoded mechanisms the opalescence of these relations cannot fully be taken into account. We are convinced that a flexible mechanism such as freely definable methods for VML metaclasses' instances and metainstances is mandatory to come up with an appropriate modeling. - Once a kind of aggregation has been modeled and implemented on the metaclass level it need not be repeated in the individual cases, e.g. between classes SECTION and SUBSECTION, between classes SUBSECTION and CHAPTER, and so on. An integrity constraint to be verified by an aggregation-metaclass method is that there are no cycles in the aggregation hierarchy, such as SECTION - SUBSECTION - CHAPTER - PARAGRAPH - SECTION, to give an example.

The approach to specialization is basically the same. To introduce our terminology, consider bibliographical entries that can be categorized into independent publications (books), dependent ones (articles), special publications (dissertations, proceedings etc.) and journals. We call an individual bibliographical entry a *generalization instance*, an individual journal and the like *specialization instances*. One real-world object is represented by different database objects. In

```
OBJECTTYPE CATSPEC_INSTINSTTYPE
    INTERFACE
    METHODS
        initHasSpec (specInst: OID): BOOL;
            //This method is applied to a generalization instance to specify that the object
            //'specInst' is one of the specialization instances.
        initHasGen(genInst: OID): BOOL;
            //Symmetrically, the target is the specialization instance, and the parameter
            //('genInst') is the generalization instance.
        ...
    IMPLEMENTATION
    PROPERTIES
        genOfObjs: {OID};          //spec. instances
        specOfObj: OID;            //gen. instances
        ...
END;
```

Figure 7: VML Code Fragment of a Metaclass's Instinsttype

VML there would be a class BIBENTRY on the one hand and classes INDEPPUB, DEPPUB, SPECPUB and JOURNAL on the other hand. The class BIBENTRY is called *generalization class*, classes such as JOURNAL *specialization classes*. The real-world features all objects being categorized have in common are modeled as the generalization instances' properties and methods. We call this principle *generalization principle*. The example is one of *category specialization*. While category specialization reflects the objects' structure, *role specialization*, on the contrast, reflects the objects' behavior, e.g. STUDENT or PATIENT are role-specializations of PERSON. A relevant fragment of a metaclass for category specialization is given in Figures 5, 6, and 7. OID ('object identifier') is the type of objects of arbitrary types.

In this context it is important that in VML instances of metaclasses can be created at runtime. Their type and the metainstances' type is part of the metaclass definition.

4 A VODAK Application Framework for SGML Documents

4.1 Modeling Issues

This section describes the core structure of a prototypical VODAK application framework for the storage of structured documents. It seems to be a straightforward option to model element types as VML classes being part of the schema. SGML attributes would be just properties. In that case, however, the requirement that DTDs must be modifiable dynamically would not be met. System shutdown every time a DTD is altered or a new one is introduced would not be acceptable.

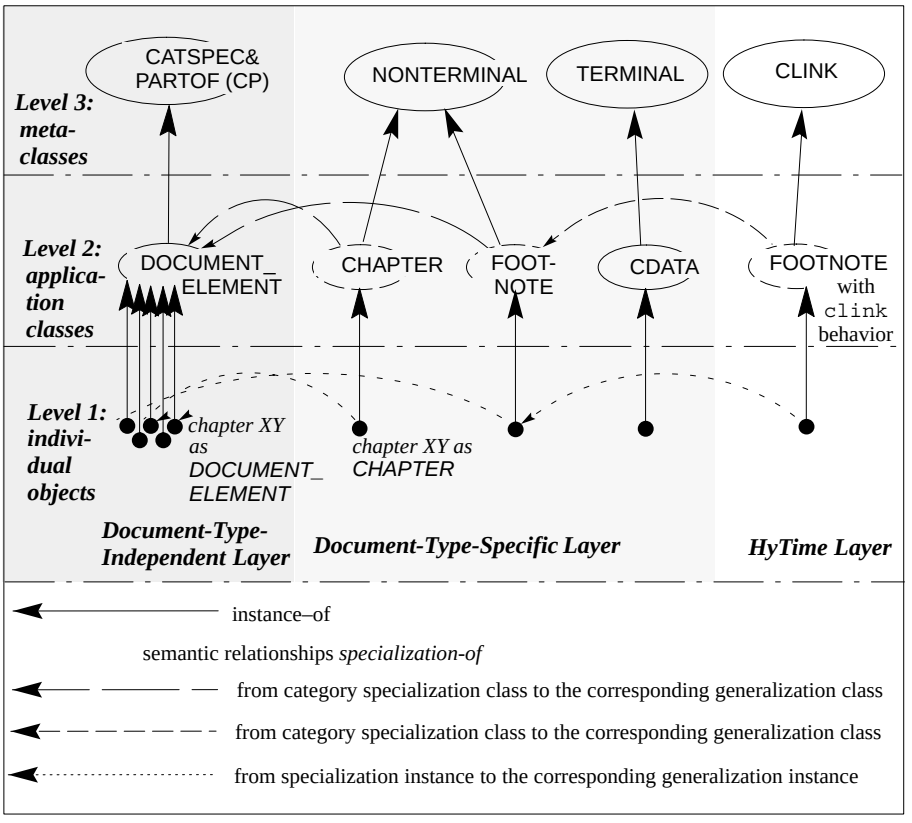

Figure 8: Overview of the Modeling

Overall Structure. We differentiate between document-type-independent features, document-type-specific ones and HyTime-specific ones. On the one hand, elements have element-type-independent or document-type-independent characteristics, e.g. the fact that they make up a hierarchical structure with other elements. On the other hand, attributes, to give an example, are element-type- or document-type-specific. Correspondingly, the schema consists of a *document-type-independent layer*, the *document-type-specific layer*, and the *HyTime layer* as in Figure 8.[3] HyTime-related issues are part of the following section. Classes are represented by ellipses. Objects that are not classes are just dots. The fact that an object is an instance of a class is displayed by a plain line arrow between them.

[3]The diagram includes the class system 'metaclass - (application) class - instances'. The rungs of this hierarchy will be referred to as levels. Levels and layers are orthogonal.

The distinction between 'document-type-specific' and 'document-type-independent' is according to the generalization principle. For every SGML-element type there is a corresponding application class in the document-type-specific layer . In the sequel, we refer to these classes as *element-type classes*. Each element-type class is a specialization class of the class DOCUMENT_ELEMENT in the document-type-independent layer. For every real-world document element there is an instance of the corresponding element-type class and another one of the generalization class DOCUMENT_ELEMENT.

Document-Type Independent Layer. The objects in the document-type-independent layer have the element-type-independent features, due to the generalization principle: The structural information is part of this layer, as well as, say, methods to navigate through the tree.

In the previous section metaclasses modeling aggregation and specialization have been described. The class DOCUMENT_ELEMENT and its instances take part in more than one semantic relationship: First, the category-specialization relationship with element-type classes as specialization classes, second, the aggregation relationship. In this particular partOf-relation parts and wholes are instances of the same class, namely DOCUMENT_ELEMENT. Because document elements may take part in the partOf-relation independent of their type, the partOf-relation is between generalization instances, due to the generalization principle. An aggregation metaclass capturing the semantics of the partOf-relation between document components cannot be used together with a metaclass for category specialization, because an object is instance of exactly one class: Inheriting features from several metaclasses could lead to unforeseeable overlappings of the metaclasses' types. Instead the metaclasses' types must be related via the subtype mechanism. A new metaclass is defined: The insttype is subtype of these metaclasses' insttypes; the same holds true for the instinsttypes. Hence, instances and metainstances have both specialization and aggregation semantics. (In Figure 8 this metaclass is called CATSPEC&PARTOF.) The special problems that occur when combining different semantic concepts are described in a forthcoming article. A relevant portion of the VML-code for this layer is given in Figure 9.

Document-Type Specific Layer. This layer bears the element-type-specific information. There is a differentiation between *terminal* and *nonterminal element types*. Terminal element types such as `CDATA` are part of ISO 8879-1986 [17] and can be used in any DTD. Hence, they are not introduced at runtime, as opposed to DTD-specific nonterminal element types. The designation 'terminal' reflects the fact that their instances are leaves of the document tree. Their processing differs from the one of nonterminal elements, e.g. because they do not have attributes according to the standard.

Nonterminal element-type classes have properties bearing the information that constitutes the element type. These properties are inherited from the metaclass NONTERMINAL and instantiated for the first time when the class is created. Likewise, the properties and methods of the element-type classes' in-

```
CLASS Document_Element METACLASS CP
    INSTTYPE
        INTERFACE
        METHODS
            getUp(): OID;
            ...
                //Method to navigate through the document tree
            getElementTypeName(): STRING;
                //Method that can be invoked for SGML elements of arbitrary types
        IMPLEMENTATION ...
END;
```

Figure 9: VML Code Fragment of the Document-Type Independent Layer

```
CLASS TERMINAL
METACLASS METACLASS
    INSTTYPE
        INTERFACE METHODS
            createElem(): OID;
                //returns the OID of newly created terminal elements,
                //e.g. instances of PCDATA
        IMPLEMENTATION ...
```

Figure 10: VML Code Fragment: Metaclass for Terminal Element-Type Classes

stances are inherited from that metaclass. The dual-model conception facilitates the creation of element-type classes without defining new types. Element-type classes are containers for SGML elements of the same element type. - In Section 2 it has been implied that SGML DTDs are essentially grammars defining the documents' structure. The semantics of document components does not follow from the element-type definitions.[4] The effect is that no element-type-specific processing is necessary. Element-type classes' instances are of the same VML type. In the database, the content of an SGML element is a list of instances of the generalization class DOCUMENT_ELEMENT. The attribute values are, according to the SGML standard, a sequence of characters. The interpretation of user-defined attribute types is not part of SGML. The code fragments in Figures 10 through 12 serve as an illustration. '|[...]|' are list delimiters. The method setContent is element-type-specific because it is only meaningful for non-terminal element-types. Furthermore, the method may check the new content's conformance to the element-type's content model. Hence, the method performs differently for different element types.

Type-to-Class Mapping. Figure 13 is a fragment of Figure 8: The boxes next to the objects indicate from which types the objects inherit their proper-

[4] An exception are SGML types `ID`, `IDREF`, `IDREFS`. These types, however, need not be used in connection with documents having a tree structure only. Therefore, they have not been introduced.

```
CLASS PCDATA METACLASS TERMINAL
    INSTTYPE
        INTERFACE METHODS
            getContent(): STRING;
            setContent (newContent: STRING): BOOL;
        IMPLEMENTATION PROPERTIES
            content: STRING;
        ...
```

Figure 11: VML Code Fragment: Class Comprising `#PCDATA` Elements

ties and methods. An instance of DOCUMENT_ELEMENT, for instance, has the properties and methods defined in the instinsttype of CATSPEC&PARTOF and the ones from the insttype of DOCUMENT_ELEMENT. The line between types within a box display the subtyping relation: CP_INSTINSTTYPE is subtype of CATSPEC_INSTINSTTYPE and of PARTOF_INSTINSTTYPE. The plain lines outside the boxes connect the types with the classes where they are defined.

4.2 Functionality and Experiences

Querying Documents. With our approach queries may refer to documents' structure. This is accomplished by introducing appropriate methods both for the document-type independent objects and the element type classes' instances. The following queries based on the document-type definition in Figure 3 serve as illustrations.

1. "Select all sections whose author is Neuhold".

   Using the VODAK Query Language VQL this query can be expressed as follows:

   ```
   Q1 :=ACCESS s
        FROM s IN section
        WHERE (s -> getAttributeValue ('AUTHOR') == 'Neuhold')
   ```

   In this query the typing of SGML elements is exploited. section denotes the object identifier of the class object corresponding to the element type `section`. In query 4 it is shown how to obtain the object identifier of a particular element-type class.

2. "Select all new terms being contained in sections selected in Q1."

   ```
   ACCESS d
   FROM d IN DOCUMENT_ELEMENT, s IN Q1
   WHERE (d-> isContainedIn (s -> categorySpecializationOf()))
              AND (d -> getElementTypeName() == 'NEW_TERM')
   ```

```
CLASS NONTERMINAL
    OWNTYPE
        INTERFACE
        METHODS
            createElementType (elemTypeName: STRING, ...): NONTERMINAL;
            ...
                    //returns the newly created element-type class
    INSTTYPE SUBTYPEOF CATSPEC_INSTTYPE;
        INTERFACE
        METHODS
            setContentModel (newContentModel: STRING);
            setAttributeModel (newAttributeModel: STRING);
            createElem(): OID;
            ...
        IMPLEMENTATION
        PROPERTIES;
            elementTypeName: STRING;
            contentModel: STRING;
            attributeModel: STRING;
            ...
    INSTINSTTYPE SUBTYPEOF CATSPEC_INSTINSTTYPE;
        INTERFACE
        METHODS
            setContent (newContent: |[OID]|): BOOL;
            ...
```

Figure 12: VML Code Fragment: Metaclass for Nonterminal Element-Type Classes

In this case the tree structure of SGML documents is referred to. Note that s is in the document-type specific layer, but d is in the generalization layer. categorySpecializationOf is used to shift between these layers. The method isContainedIn makes use of documents' tree semantics.

3. "Select the first elements of the documents."

```
ACCESS d -> getFirst()
FROM d IN DOCUMENT_ELEMENT
```

This query makes use of document contents' ordering. From the DTD we can infer that the elements selected represent elements of type `section`. getFirst is a method to return the first content element.

4. "Select all element-type classes representing element types with content model '`paragraph*`'."

```
ACCESS n
FROM n IN NONTERMINAL
WHERE n.contentModel == 'PARAGRAPH*'
```

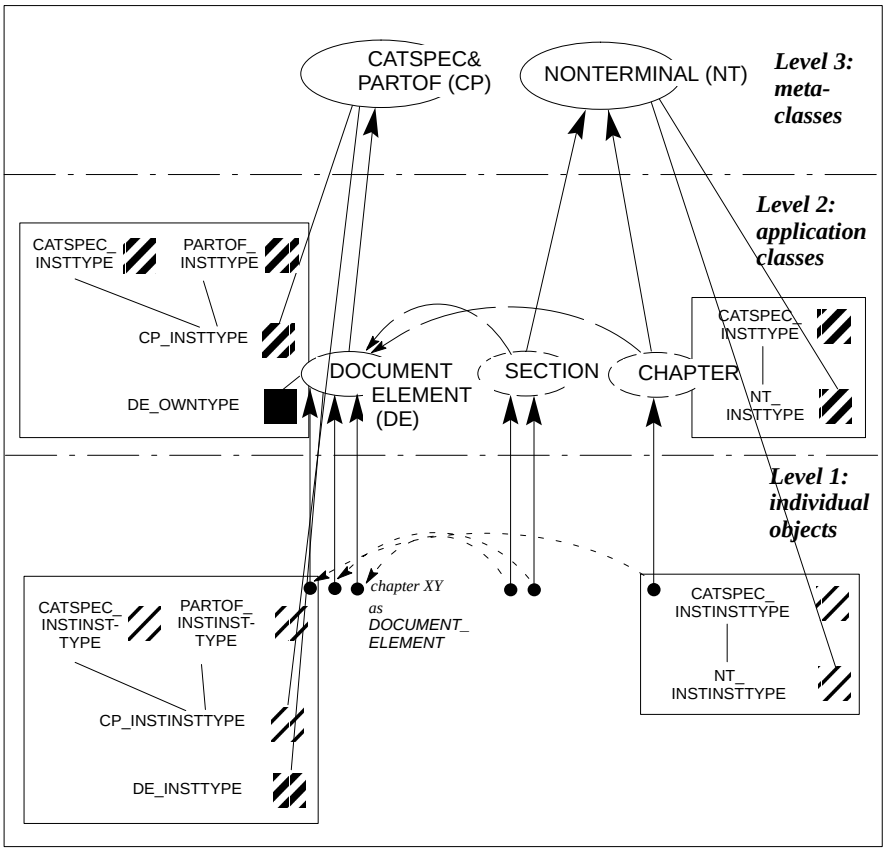

Figure 13: Dual Model: Document-Type Independent and Document-Type Specific Layers

> The query illustrates that in our framework DTDs can be accessed in a natural way.

We claim that with this approach document querying is as expressive as with other approaches currently discussed in literature [11, 26]. The sample queries could now be optimized by using application-specific semantics [3].

Concurrency Control. Extended functionality, as compared to previous approaches to document handling, is our objective. Multi-authoring (i.e. multi-user mode) and versioning have been mentioned. - In VML there exists the possibility to brace a sequence of operations to a transaction. Then the DBMS inter alia ensures that no interleaving with other operations occurs. Multi-authoring need not be realized as part of the database application, but instead built-in features can be applied. The code fragment in Figure 14 is part of a sequence

```
BEGIN_TRANSACTION
        BODY := NONTERMINAL -> createElemType('Body', ...);
        BODY -> setContentModel('(Paragraph)*');
        body := BODY -> createElem();
        ok := body -> setContent ({(1, para1), (2, para2)});
        ...
IF (ok)
        COMMIT_TRANSACTION
ELSE
        ABORT_TRANSACTION
```

Figure 14: Example Using VML Transactions

of operations generating an instance of the DTD in Figure 3. The meaning of BEGIN_TRANSACTION, COMMIT_TRANSACTION and ABORT_TRANSACTION is canonical. BODY is the object identifier of an instance of NONTERMINAL, i.e. an element-type class. The method createElemType creates a new instance of the class NONTERMINAL. Because in VODAK classes and metaclasses are first-class objects they can receive method calls. The second operation sets the content model of the element-type class BODY to '(Paragraph)*'. Now an instance of BODY can be created: The method createElem actually creates two objects: An instance of the target-element-type class and the corresponding generalization instance. The first one is returned; the variable body is instantiated with it. In the next step, the content of body is instantiated with a list of paragraphs: para1 and para2 are instances of DOCUMENT_ELEMENT that must have been created before. ok is a variable of type BOOL. setContent returns TRUE iff it executed as foreseen.

Versioning. For versioning it is necessary to break down the authoring process into so-called tasks. As a rule, each task's end corresponds to a version of the document, the result of the task. With our approach granularity for versioning is on the SGML element level: If within a task a small portion of the document is altered, it is not necessary to generate a copy of the entire document, but only copies of the elements that have been modified. They are generated automatically by operations being part of the versioning metaclasses if within the current task a copy of the element being modified has not yet been generated. Hence, the interface of edit-operations (e.g. setContent) needs not be altered. On the other hand old document versions can be read and new versions can be derived from them. In the world of non-versioned objects, for example, there is a method printDocument. In the world of versioned objects it has a counterpart printDocument (task: taskType) displaying the state of the document at the end of the relevant task.

Coupling with SGML Parser. To insert documents as a whole into the system, the SGML parser ASP ("Amsterdam Parser") [28] is coupled with the database application. The ASP has been extended to invoke the methods creating the database objects that represent the logical document components. This

is already part of the parsing process. On the other hand, the parser also checks the document's conformance to its DTD. Because after the parsing process it is known that the document is correct certain verifications can be omitted that are part of the operations in the general case. For instance, one axiom states that the aggregation relation is cycle-free. This is checked within the corresponding operation of the metaclass PARTOF every time a new relation is introduced. Because of the documents' correctness after parsing, i.e. the document has a tree structure, checks of this kind can be omitted when inserting documents the way just described. This is advantageous for performance reasons. The metaclass PARTOF provides another version of that operation without that check. It is to be used within the coupling with the ASP. This is feasible because of the extensibility of the data model, i.e. the freely definable methods in metaclasses. Application-specific knowledge is not only necessary to model semantic relationships. It is also advantageous with regard to efficiency.

Experiences. Setting up the hierarchical structures representing documents in the database naturally results in an overhead for storage with regard to time and size. Documents stored in OODBMSs according to their tree structure naturally occupy more disk space than stored as a file: First experiments indicate a growth in size by a factor of ten for "structure-only" documents as compared to an ASCII file representation. Because "normal" documents exist, to a large degree, of text, which does not lead to any enlargement of the tree, the average factor to be expected is substantially lower.

Previously, our applications have been based on an SGML document base which, in turn, is based on a relational database system. It seems that updates for small document sets (i.e. both number and size of documents are small) are comparable in duration to our current system. For large document sets relational systems reportedly tend to cause problems. The reason is that systems based on this technology have to stick to a fixed schema. Hence, relational tables inevitably grow. This effect is not to be expected with a dynamic approach such as the one presented in this article. Besides that, with the relational paradigm access requires a large number of join operations. Thus one would expect that an object-oriented system behaves superior already for small document sets. With our implementation this experience has actually been made. In addition, frequent access operation schemes can easily be accelerated both in a general or SGML-specific manner. An example of an access scheme with optimization potential would be the search for the content of elements of a particular element type. Another example are pointers from document elements to the root element speeding up that kind of navigation.

5 Extending the Framework with HyTime Features

The objective of a significant portion of the concepts and technologies that are commonly subsumed under the term 'digital libraries' is the processing of multimedia documents. For multimedia documents the internal structure of the documents must be taken into account, as explained in Section 1. At a naive level of analysis, the HyTime standard is seen as an extension of SGML to deal with documents' hypermedia content. However, a more differentiated view is necessary because SGML documents may already be multimedia documents with hyperlinks. The objective of the HyTime standard is to capture the document elements' semantics. With SGML, it is only the structure of documents that can be defined. The element's interpretation is left to the reader and, hence, is not uniform in general. With hypermedia documents where document components need actually be processed this is not sufficient. Thus, the HyTime standard essentially is a list of element-type definitions together with an informal, but binding specification of these elements' processing semantics. Element-type definitions with a fixed semantics are referred to as *architectural forms*. The architectural forms provided by the HyTime standard are classified into modules according to their function: in addition to a basic module there are the measurement module and the location module, whose architectural forms can be used to identify arbitrary locations in the documents or the presentation space, e.g. the seventh to eleventh word, the hyperlinks module, whose architectural forms are templates for link structures, a scheduling module, whose elements are indeed useful for (time-dependent) multimedia content, and a rendition module (that is of minor importance in this context). In most cases, in application-DTDs HyTime architectural forms are not directly included, but instead refinements of them are introduced. Refinements of architectural forms are possible in two ways,

1. by introducing additional attributes,
2. by confining the range of content or attribute types.

For illustration purposes consider the example taken over from [2] with slight modifications. The name of the architectural form `clink` (see Figure 15) is short for 'contextual link'. An instance of `clink` is a reference together with content, such as footnotes (see the element-type definition being part of an application DTD in Figure 16). With regard to '`(%HyBrid;)*`' in this content it suffices to know that '`#PCDATA`' is a specialization of it (cf. item 2 from above). It may be helpful to have a reference from a footnote to the following one. To this end, the attribute `following` that is not part of the architectural form `clink` has been introduced (cf. item 1). The attribute `HyTime` in Figure 16 specifies that `footnote` is a specialization of a HyTime architectural form, namely `clink`.

```
<!element   clink       -- Contextual link --
                        (%HyBrid;)*>
<!attlist   clink       HyTime      NAMEclink
                        id          ID
                        linkend -- Link end --
                                    ...
                                    IDREF   ...>
```

Figure 15: Example of HyTime Architectural Forms

```
<!ELEMENT   footnote#PCDATA>
<!ATTLIST   footnoteHyTime          NAMEclink
                        id          ID
                        linkend IDREF
                        following   ID>
```

Figure 16: Example of Element-Type Definition in HyTime Application DTD

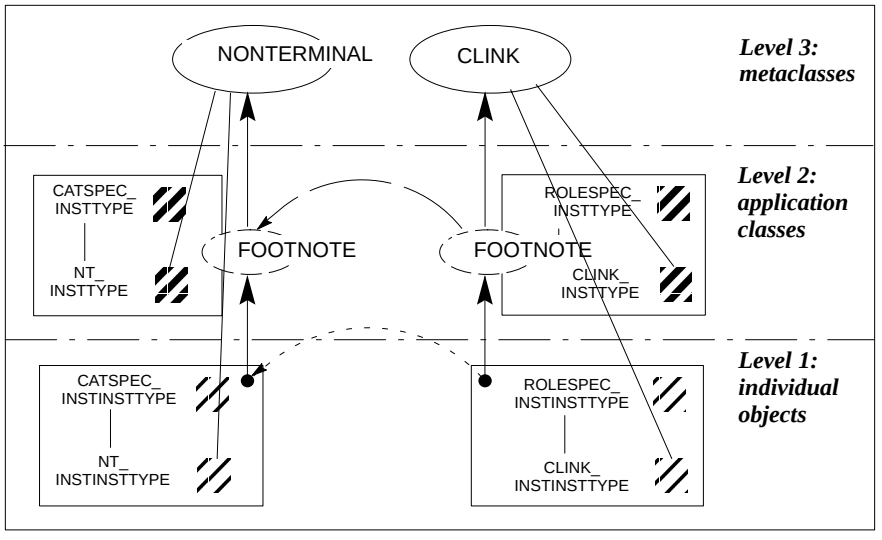

Figure 17: Dual Model: HyTime Layer

```
CLASS CLINK
    OWNTYPE
        ...
    INSTTYPE SUBTYPEOF ROLESPEC_INSTTYPE;
        INTERFACE METHODS
            createSpecInstance(): OID;          ...
        IMPLEMENTATION      ...
    INSTINSTTYPE SUBTYPEOF ROLESPEC_INSTINSTTYPE;
        INTERFACE METHODS
            getReferencedElement(): OID;        ...
```

Figure 18: VML Code Fragment: Metaclass for Architectural Form `clink`

With regard to the realization of architectural forms the following observation is fundamental. The (HyTime-)aspect is a role specialization of the corresponding SGML element. However, a metaclass ROLESPEC for role-specialization classes and instances cannot be applied in a straightforward way: Analogously to SGML element-type classes the corresponding role-specialization classes need be created dynamically. Hence, they themselves as well as their instances need to inherit property- and method definitions from their metaclass. A metaclass provides both the role-specialization semantics and the semantics of a particular HyTime architectural form. Again, the type definitions are related via subtyping (see Figure 17). A relevant fragment of the metaclass corresponding to `clink` is contained in Figure 18. getReferencedElement is a fairly simple example of a method reflecting the semantics of the architectural form `clink`. To summarize, an SGML element with HyTime semantics is represented by three database objects: A generalization object bearing the element-type-independent semantics, a specific one with element-type-specific features and one with properties and methods modeling the HyTime semantics.

6 Embedding the Application Framework in Various Scenarios

In this section an integral scenario is described that may rely on the database application framework presented in the previous section. Extensions such as the ones that will be described are meaningful in order to offer typical services of digital libraries. Some of these extensions have been, others are currently realized as part of the project HyperStorM ('Hypermedia Document Storage and Modeling') at our institute.

6.1 Coupling with Information-Retrieval Systems

Combining content orientation with structure orientation in queries on documents is an issue of current research. Examples for queries of that kind are 'Select all documents whose introduction deals with the information highway.'

or 'Select all section headers about the World Wide Web.'. On the one hand, we are aware of systems being able to handle documents of arbitrary structure [9]. However, instead of content-based search there merely exists the possibility to search the text for patterns. On the other hand, with state-of-the-art information-retrieval systems a fixed set of document types is allowed at best. In [13], to give an example, there is a limited set of document-component types such as paragraph or section which may be referred to in queries on documents' content.

When coupling information-retrieval systems with database applications there is a principal distinction between loose coupling and tight coupling. With tight coupling the information-retrieval functionality is part of the database application. With loose coupling data is not shared. There is merely communication on the documents between the database application and the information-retrieval system. Tight couplings in general are more efficient and less space-consuming. However, with tight coupling one is confined to a certain approach towards information retrieval.

Recently, at our institute a loose coupling between our database application and the information-retrieval system INQUERY [10] has been realized. INQUERY index structures are set up for elements of certain types. These element types can be chosen arbitrarily. If an element's content is modified or an element is created or deleted the index structure is altered accordingly. It is not advantageous to set up INQUERY index structures for elements with little textual content (less than 30 words as a rule) because meaningful content-based queries are not possible. On the other hand consider queries referring to document elements with INQUERY index structures existing only for their components. Experience shows that relatively good results are obtained by sending the query to the components and combining the results, e.g. by using the maximum-function or sum-function, whatever is appropriate.

6.2 Combining Documents' Content with Knowledge Bases

With the extremely large amount of data to be administered in digital libraries the assumption that access can be accomplished without additional knowledge is not realistic. Consider the query 'Select the descriptions of all German universities.'. In order to evaluate this query the names of these institutions are first retrieved from a database. These names are inserted into the query to the document base. Furthermore, independent from a particular application scenario the knowledge base may also contain fuzzy terminological knowledge that, analogously, may be called upon to evaluate document queries [21]. - To model knowledge semantically rich data models are advantageous. The possibility to make a broad variety of modeling primitives available has been an objective when developing VODAK.

6.3 Handling HTML Documents

While the primary structure of conventional documents is a tree, as described in Section 2, hypertext documents have a graph structure. The World-Wide Web (WWW) [23] is such a hypertext structure whose nodes may be physically distributed without restrictions.[5] The nodes are conformant with an SGML-DTD, the HTML-DTD ('Hypertext Markup Language'). In addition to this DTD there is a universal naming scheme for documents so that the WWW can be browsed as one large document. With HTML, SGML concepts are, to a degree, misused. The strict distinction between logical and layout structure has been dissolved because in this special context it is supposedly more convenient. E.g. there is an element type 'Line Break', and element type 'Paragraph' has attributes 'align', 'indent' etc. Another interesting feature is that actions, i.e. how to process the document together with some user entries, can be specified. In this context it is important to notice that HTML documents are SGML documents, and, thus, can be stored using our database-application framework. When accessing WWW documents through a database typical database-integration problems can be encountered. Namely, WWW servers can be seen as external multimedia databases. According to [19] one can distinguish two different stages in database integration, the *syntactic transformation phase* and the *semantic integration phase*. In the syntactic transformation phase the external databases' representation is transformed so that it can be accessed and manipulated by the database serving as the integration platform. In our case, this is VODAK. To this extent two approaches may be considered.

- With regard to the first approach it is assumed that a relatively small number of external documents is accessed frequently. Under this assumption physically importing the document is a reasonable approach. A WWW document can in principle be considered as a pair (location identifier, SGML document as file). Hence, importing the document into the database only needs internet access via the location identifier and consecutive parsing of the SGML file. Parsing is possible due to the coupling with the ASP. However, a restriction in this context is the fact, that the majority of HTML documents is not conformant to the DTD. Presentation of these documents in principle is possible, storage according to the logical structure, on the other hand, is subject to restrictions.

- With regard to the second approach there is the assumption that large numbers of documents will be queried rather seldomly. In that case physical import does not make sense. It is necessary to provide an appropriate view on the documents according to their virtual logical representation.

[5] The WWW is another example for the phenomenon that the boundary between intra- and inter-document processing is not clean-cut. On the one hand the WWW can be seen as one large document, from another perspective the individual nodes are documents in their own right.

This can be facilitated by allowing parsing operations on the documents in query evaluation [4].

Once access to WWW documents is provided in one way or the other, the semantic integration phase begins. Several ways of semantic enrichment and integration are feasible with regard to WWW documents: The most obvious option is to support the particular HTML-semantics. In particular, supporting the semantics of HTML's world-wide links, e.g. by resolving them explicitly to physically imported documents, is conceivable. Furthermore, in many cases WWW documents are in accordance with some implicit editorial conventions. To make this implicit structure and semantics explicit, e.g. by transforming the documents to more application-specific document-type definitions, is subject to current research [5]. Consider conference proceedings in HTML format available only via WWW (cf. [1]). It might be necessary to transform these proceedings from HTML to proceedings conformant to a DTD in another format. A third item that may be subject to further work is overlapping information in the WWW. Different approaches for providing an integrated view (on this potentially contradicting information) would have to be developed. Schema integration is a related issue in the database area [19].

Different issues with regard to the integration of WWW document bases in an SGML database have been discussed. They reflect a particular kind of architecture: The database is on the consumer side. This might be particularly relevant to improve access for special interest groups in the WWW. It may be known that they regularly access a certain fragment of the WWW. Improving access has two facets: First, by caching frequently accessed documents access is substantially accelerated. Second, by enriching the documents syntactically and semantically, retrieval capabilities are enhanced in a nontrivial way. Such a "consumer server" will be used by several users. To this end, DBMS functionality is mandatory.

Another topic remaining to be explored is the use of SGML database technology on the document producer side. This will be particularly significant with a large number of users contributing to a WWW documents' "archive". Without database technology this can only be accomplished by an archiver's manual intervention. Otherwise consistency of the archive is at stake. Conventional database updates would correspond to the insertion of documents that are completely new.

6.4 Coupling with DFR-Archive

With the current version of the database application framework documents are apportioned to pools. Pools are merely sets of documents. For some applications this classification is appropriate. To meet the requirements arising in connection with digital libraries a more sophisticated approach is imperative. The DFR-standard is a platform to define the documents' classification. There

is a primary structure, which is hierarchical: The root is referred to as *DFR root group*, the internal nodes are *DFR groups*. A document or a DFR group may directly be contained in one DFR group. Besides that, there are DFR reference objects that model a containedIn-relationship between DFR groups that are not directly related in the tree. Names and attributes for the individual DFR objects are freely definable thus reflecting the concrete application semantics. Documents' internal structure likewise is not part of a DFR specification. A DFR archive that has been realized on top of VODAK is described in [22]. - The combination of a DFR archive with our database application for SGML documents is advantageous if a large number of SGML documents is to be administered. Consider a DFR structure for travel guides classifying documents by the location they describe. Furthermore, assume that these travel guides are SGML documents having an introduction. Then queries such as, say, 'Select the introductions of all documents about Heidelberg' can be formulated and evaluated.

6.5 Architecture

The disposition of the individual components of the digital library for structured documents we envision is displayed in Figure 19. Arrows reflect the flow of data between the components. In more detail, arrows labeled with A stand for documents' insertion into the document base. Arrow B reflects the fact that SGML documents can be generated from the database content in a straightforward way. - The MultiMedia Forum is an interactive online journal published by our institute. An issue of the journal consists of a number of articles, but the reader is free to look only at the documents he is interested in. Articles are SGML documents conformant to an SGML-DTD with inter-document references. From a different perspective, it is the entire issue of a journal that might be seen as a document in its own right. Then these links become intra-document references.

7 Conclusions

Starting point of this article has been the observation that both a documents internal structure as well as the relations between documents should be properly reflected when documents are stored in digital libraries. The phenomenon that the dividing line between inter-document relationships and intra-document relationships is not clean-cut has been hinted at by means of examples, such as the WWW. Describing documents according to their internal structure is advantageous with regard to the various services offered by digital libraries. Within our database application framework documents of arbitrary types can be handled. Querying documents according to their structure is an issue currently attracting researchers' attention [6, 11, 8]. The queries given there can also be formulated using the VODAK Query Language VQL. Some sample queries

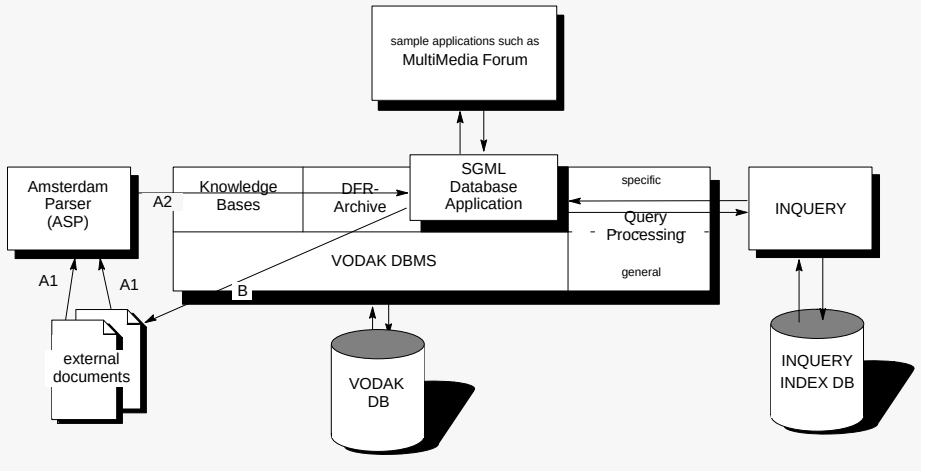

Figure 19: Proposed Architecture for SGML-based Digital Library

have been formulated to sketch the expressive power of VQL. We have briefly explained how to amalgamate the system with the extension toward HyTime semantics. Besides that, the coupling with other modules for enriched functionality has been explained. The systems that have been mentioned in this context are an information-retrieval system, knowledge bases and a DFR archive. An interface to the WWW has also been discussed.

The main issued of this chapter has been that storage of SGML documents of arbitrary types is possible with VODAK. The VODAK data model differentiates between types and classes. If the necessary types are defined, classes may be generated at runtime. This feature is exploited both for the dynamic creation of element-type classes and for role-specialization classes bearing HyTime semantics. In the application framework there exist metaclasses reflecting the semantics of the different HyTime aspects. The corresponding types are combined using the subtyping mechanism of VODAK to provide the necessary types for the HyTime-oriented classes created for the different applications. Using an OODBMS has the general advantage that the application semantics - in this case the SGML and HyTime semantics - are part of the database.

Leaving aside the completion of the integration of the components mentioned in Section 6, an issue that may be part of future work is to improve the efficiency of retrieval operations. With documents stored in accordance with their hierarchical structure optimization of tree structure retrieval is an important topic. There will also be even more potential to optimize access by including the other system components that will contribute to an integrated digital library.

Acknowledgements. We thank Peter Muth, Thomas Rakow and Marc Volz for their comments on an earlier version of this article and Ute Sotnik for

correcting mistakes.

References

[1] K. Aberer. Demand-driven database integration for biomolecular applications. In *Electronic Proceedings of the Meeting on the Interconnection of Molecular Biology Databases, WWW page http://este.darmstadt.gmd.de:5000/ aberer/MIMBD.html.* Stanford University, August 1994.

[2] Karl Aberer, Klemens Böhm, and Christoph Hüser. The prospects of publishing using advanced database concepts. In Christoph Hüser, Wiebke Möhr, and Vincent Quint, editors, *Proceedings of Conference on Electronic Publishing*, pages 469–480. John Wiley & Sons, Ltd., April 1994.

[3] Karl Aberer and Gisela Fischer. Semantic query optimization for methods in object-oriented database systems. accepted for publication in Proceedings of International Conference on Data Engineering 1995, 1994.

[4] S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In R. Agrawal, S. Baker, and D. Bell, editors, *Proceedings of the International Conference on Very Large Data Bases*, pages 73–84. VLDB Endowment, 1993. Dublin, Ireland.

[5] E. Akpotsui and V. Quint. Type transformation in structured editing systems. In C. Vanoirbeek and G. Coray, editors, *Proceedings of Conference on Electronic Publishing*, pages 27–42. Cambridge University Press, 1992. Lausanne, Switzerland.

[6] Paula Angerstein. Sgml queries, December 1992. Handout for the Session on SGML Query Languages at the SGML'92 Conference.

[7] J. Banerjee et al. Semantics and implementation of schema evolution in object-oriented databases. *Proceedings ACM SIGMOD*, 16(3):311–322, 1987.

[8] G.E. Blake et al. Text / relational database management systems: Harmonizing sql and sgml. In *Proceedings of the First International Conference on Applications of Databases.* Lecture Notes in Computer Science, Springer Verlag, June 1994.

[9] F.J. Burkowski. Retrieval activities in a database consisting of heterogeneous collections of structured text. In N. Belkin, P. Ingwersen, and A.M. Pejtersen, editors, *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–125. ACM Press, 1992.

[10] J.P. Callan, W.B. Croft, and S.M. Hardig. The inquery retrieval system. In *Proceedings of the Third International Conference on Database and Expert Systems Application*, pages 78–83. Springer Verlag, 1992.

[11] V. Christophides et al. From structured documents to novel query facilities. In *Proceedings ACM SIGMOD*. ACM Press, May 1994.

[12] J. Conklin. Hypertext: An introduction and survey. *IEEE Computer Magazine*, pages 17–41, September 1987.

[13] W.B. Croft, L.A. Smith, and H.R. Turtle. A loosely-coupled integration of a text retrieval system and an object-oriented database system. In N. Belkin, P. Ingwersen, and A.M. Pejtersen, editors, *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 223–232. ACM Press, 1992.

[14] W.B. Croft and D.W. Stemple. Supporting office document architectures with constrained types. In *Proceedings ACM SIGMOD*. ACM Press, May 1987. San Francisco.

[15] M. Halper et al. Integrating a part relationship into an open oodb system using metaclasses. In *Proceedings of Third International Conference on Information and Knowledge Management (CIKM'94)*. ACM Press, November 1994.

[16] Information technology - text and office systems - standardized generalized markup language (sgml), 1986. ISO 8879-1986 (E).

[17] Information technology - text and office systems - office document architecture (oda) and interchange format, 1989. Part 2, Document Structures.

[18] W. Klas, K. Aberer, and Erich J. Neuhold. Object-oriented modeling for hypermedia systems using the vodak modeling language (vml). In A. Dogac, T. Ozsu, A. Biliris, and T. Sellis, editors, *Advances in Object-Oriented Database Management Systems*, NATO ASI Series. Springer Verlag Berlin Heidelberg, August 1994.

[19] W. Klas et al. *Object-Oriented Multidatabase Systems*, chapter Database Integration Using the Open Object-Oriented Database System VODAK. Prentice Hall, 1994.

[20] W. Klas et al. Vml - the vodak model language version 4.0. Technical report, GMD-IPSI, October 1994.

[21] M. Kracker. A fuzzy concept network model and its application. Technical Report 585, GMD-IPSI, October 1991. St. Augustin.

[22] F. Moser and T.C. Rakow. Database support for the access towards an open and multimedia archive. In *GI-FG Databases, Fall Workshop*, September 1993. in German, Jena.

[23] K. Obraczka, P.B. Danzig., and S.-H. Li. Internet resource discovery services. *IEEE Computer*, 26(9), 1993.

[24] L. Rostek, W. Möhr, and D. Fischer. Weaving a web: the structure and creation of an object network representing an electronic reference work. *Electronic Publishing*, 6(4):495–505, 1994.

[25] K. Shoens et al. The rufus system: Information organization for semi-structured data. In R. Agrawal, S. Baker, and D. Bell, editors, *Proceedings of the International Conference on Very Large Data Bases*, pages 97–107. VLDB Endowment, 1993. Dublin, Ireland.

[26] B. Subramanian et al. Querying lists and trees: A language and some optimizations. accepted for publication in Proceedings of International Conference on Data Engineering 1995, 1994.

[27] Klaus Süllow et al. Multimedia forum - an interactive online journal. In Christoph Hüser, Wiebke Möhr, and Vincent Quint, editors, *Proceedings of Conference on Electronic Publishing*, pages 413–422. John Wiley & Sons, Ltd., April 1994.

[28] 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.

[29] J. Zobel, J.A. Thom, and R. Sacks-Davis. Efficiency of nested relational document database systems. In G.M. Lohmann, A. Sernadas, and R. Camps, editors, *Proceedings of the International Conference on Very Large Data Bases*, pages 91–102. VLDB Endowment, 1991. Barcelona, Spain.