Arbeitspapiere der GMD
GMD Technical Report

Klemens Böhm, Karl Aberer

Amplifying the Scope of Document Handling: An Object-Oriented Database Application Framework for Structured Document Storage

GERMAN NATIONAL RESEARCH CENTER FOR INFORMATION TECHNOLOGY

Addresses of the authors:

Klemens Böhm
GMD-IPSI
Dolivostraße 15
D-64293 Darmstadt

Email: kboehm@darmstadt.gmd.de
Phone: ++49/6151/869-963
Fax: ++49/6151/869-966

Karl Aberer
GMD-IPSI
Dolivostraße 15
D-64293 Darmstadt

Email: aberer@darmstadt.gmd.de
Phone: ++49/6151/869-935
Fax: ++49/6151/869-966

Address all correspondence to:

Klemens Böhm
GMD-IPSI
Dolivostraße 15
D-64293 Darmstadt

Email: kboehm@darmstadt.gmd.de
Phone: ++49/6151/869-963
Fax: ++49/6151/869-966

Abstract

With standard database approaches to document storage the document-file objects either are left intact, or one general document structure is assumed. In the first case, full database functionality is not achieved. With regard to the second case we claim that different document types are to be treated differently. – The Standard Generalized Markup Language (SGML) is a means to identify logical document components. The generic document structure is defined by a set of production rules, a so-called document-type definition (DTD).

Using the object-oriented database management system VODAK the design of the meta layer of an application framework to store SGML documents in accordance with their structure is described. Multi-user mode, versioning and sharing of document components are considerably facilitated by appropriate fragmentation of the documents in the database. Using an OODBMS has the advantage that SGML semantics can be factored out from applications and can be integrated into the underlying system. With our approach arbitrary SGML DTDs can be communicated to the system at runtime without system shutdown. In this article we put some emphasis on describing the impact of the data model on the database application's structure.

Keywords: SGML, Structured Documents, Object-Oriented Databases, Extensible Data Model, Dual Model, Metaclasses

1 Introduction

In parallel to the exploding complexity of achievements in the fields of engineering and natural sciences size and amount of textual documents have increased in an extreme way. The critical point that documents are becoming too large for one person to handle has long been reached. Conventional systems for document storage either leave the document-file objects intact, or documents are fragmented always in the same way, independently of their structure. In order to optimize document storage, however, one conception is too monolithic. A more differentiated approach takes into account the following points:

For Shoens et al. [Sho+93] textual documents are examples of semi-structured data. We claim that

between different document types[1] there exist considerable differences: From our point of view, encyclopedic biographical articles, for instance, is structured data; novels, to give an example from the other end of the spectrum, are weakly-structured data as a rule. From another perspective, the document size is important. A document that still can be entirely cached can be treated differently than bigger ones. Third, the document-development process has some relevance: Consider the scenario of more than one author working on the document or more than one person taking part in the publishing process. In that case, document storage must be different from processes for which the involvement of only one person can be assumed. Besides that, a differentiation can be made by means of the relative frequency of update operations: In some cases a static view is sufficient, in other cases numerous iterations in the authoring process are foreseeable. Another issue impinging on document storage is the degree of interdependency between documents. If frequently parts of documents are contained within other documents, support by the storage layer is advantageous. Finally the way information is retrieved is a criterium for document storage: It is a difference whether the person querying the document base knows the documents he is searching for or whether he is looking for particular information in a document pool. In the first case, as a rule, syntactic criteria are known by which the documents can easily be identified without uncertainty (e.g. 'Document was written by Klemens Böhm.' – assuming that documents are attributed with their author's name). In the second case the degree of the search algorithms' sophistication differs: While from a scientific point of view approaches based on pattern matching are essentially interesting only with regard to performance, there are more fundamental difficulties if search is really content-based: For example, the search result may depend on the documents' fragmentation.

The documents we are looking at in this article are highly structured, very big and are repeatedly edited by several persons. Typical examples are documentations for sophisticated engineering applications or encyclopediae. Conventional system designs lead to a number of difficulties for that kind of documents. By leaving document-file objects intact multi-user mode is not supported: Granularity is on the document level. Concurrently authoring different fragments of the same document is cumbersome. Besides that, if versioning was supported in each task a copy of the entire document would be generated even if only a small fragment of the document had been modified. – On the other hand, fragmentation of documents in database applications, as practised so far, has the following

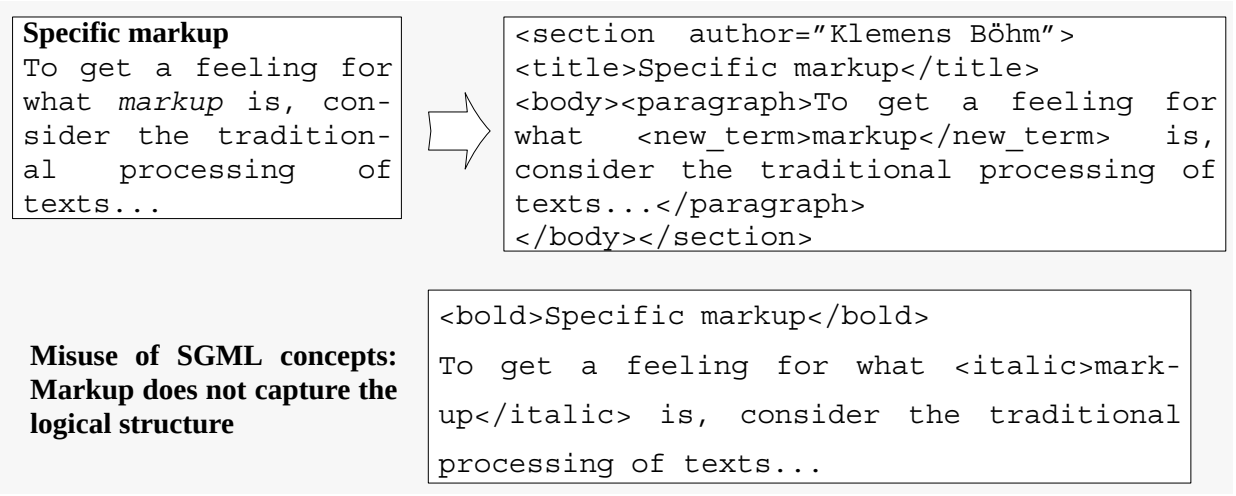
1. At a naive level of analysis, examples of document types are letter, biography, novel.

shortcomings: The question how documents are fragmented in the optimal way cannot be generally answered. It is either supposed that a generic document structure exists, or only a particular document type is supported. Furthermore it is assumed that document types do not change over time. If the system is based on a generic document type, this view might actually be sufficient. But it does not suffice for more sophisticated document handling [AkQ92].

The assets of SGML ('Standard Generalized Markup Language') [ISO86, Her90] become most evident with the kind of document we are aiming at in this article. With SGML the logical structure of documents of arbitrary types can be described. In a nutshell, this is accomplished by identifying logical document components such as sections, subsections, definitions, examples within the document. The Document Storage Agent (D-STREAT) stores SGML documents using the object-oriented database management system (OODBMS) VODAK [Kla+93, KAN93]. Those logical document-component types correspond to classes in the database. To be capable of handling arbitrary document types and of incorporating document-type change into the system these classes are created dynamically and may 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 our approach the decision how the documents are fragmented in the document base is left to the designer of the DTD. In this article the kernel structure of D-STREAT is described. Two other facets of D-STREAT, dynamic DTD handling and the impact on query formulation, have already been discussed in [ABH93].

Using an OODBMS, the SGML semantics is part of the database. One effect is that the application interface provides SGML-specific operations rather than primitive ones. Besides that, it is the groundstone for query optimization as well as semantic concurrency control (not addressed in this article). We will illustrate how VODAK concurrency control can be applied in this context, and sketch how versioning is integrated into the system.

The structure of this paper is the following: The next section contains an overview of the key concepts of SGML. The VODAK Modeling Language (VML) is reviewed in Section 3. It serves as the framework on which our database application D-STREAT is based. Its description is in Section 4. In Section 5 we review related work. The last section contains concluding remarks.



1: SGML Example - Markup

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 2 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

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

2: SGML Example - Fragment of a Document Type Definition

STRING. Sections have an attribute `author` of type `CDATA`. 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 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 2 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` being an example of a *terminal element type* contains textual data. – Aside from SGML there is the standard ODA [ISO89] and various proprietary formats to describe documents' logical structure. The concepts that are discussed in the sequel can quite easily be applied to those 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.

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 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 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 introduced 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 proper-

ties 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.

Semantic Relationships. A fundamental reason why metaclasses are in use is to model semantic relationships between classes. Examples of semantic relationships are *aggregation ("partOf")* and *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 [Ban+86]). However, semantic relations have a variety of facets. Furthermore, some facets, which are called *dimensions* in [Hal94], 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 needs 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 VML there would be a class BIBENTRY on the one hand and classes INDEPPUB, DEPPUB, SPECPUBL 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*.

Furthermore, 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

This section describes the core structure of D-STREAT, 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.

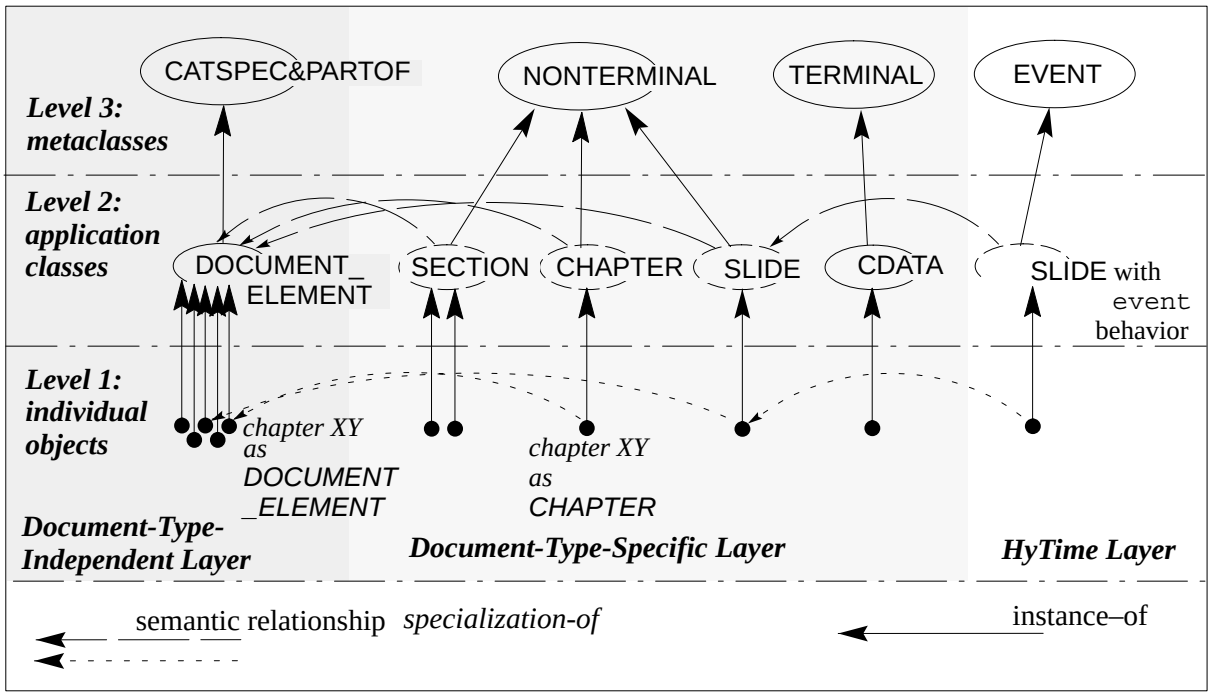
Overall Structure. We differentiate between document-type-independent features, document-type-specific ones and HyTime-specific ones[2]. Correspondingly, the schema consists of a *document-type-independent layer*, the *document-type-specific layer*, and the *HyTime layer* as in Figure 3.[3] Classes are 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. An arrow of kind ◄—— ——— — is from a specialization class to the corresponding generalization class, an arrow of kind ◄- - - - - - - - - - is from a specialization instance to the corresponding generalization instance.

The distinction between 'document-type-specific' and 'document-type-independent' is according to the generalization principle. In the document-type-specific layer an application class corresponds to every SGML-element type. In the sequel, we refer to these classes as *element-type classes*. An element-type class is a specialization class of the class DOCUMENT_ELEMENT in the document-type-independent layer. So 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 of the document trees is part of this layer, as well as, say, methods to navigate through the tree.

---

2. The HyTime standard is an extension of SGML associating well-defined semantics with SGML-document elements. An objective of HyTime is, generally speaking, to handle multimedia documents. Figure 3 contains an element-type class SLIDE to illustrate that document content need not always be in textual form. `event`, to explain the figure, may be a time interval during which the slide is displayed. This could be captured by a method display of EVENT's metainstances displaying the slide at the time interval specified. In this context it suffices to know that extending the system to deal with HyTime is straightforward. On the other hand, we do not see how the HyTime semantics could easily be integrated into a file-based system.
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.

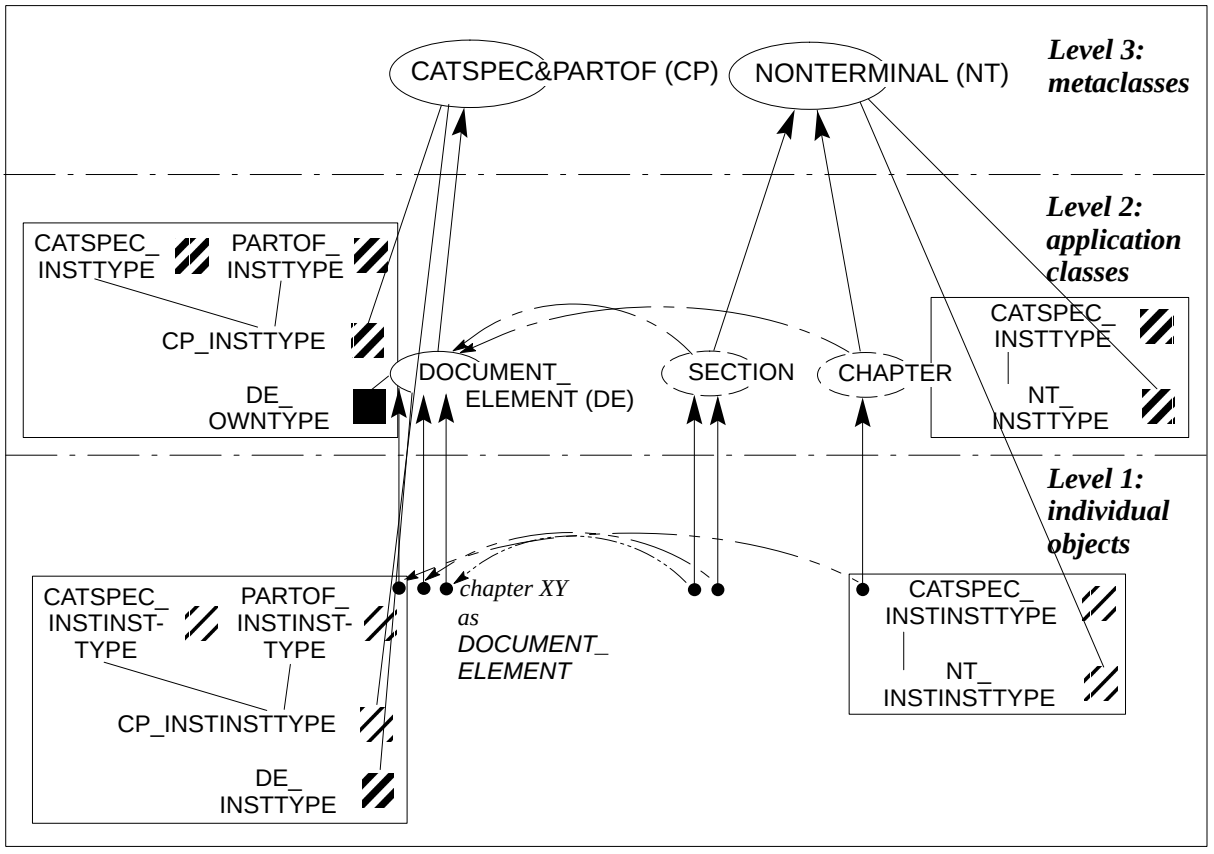


3: D-STREAT - Overview

In the previous section metaclasses to model 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 3 this metaclass is called CATSPEC&PARTOF.) At the moment we are working on the integration of versioning semantics in a similar way. The special problems that occur when combining different semantic concepts are described in a forthcoming article.

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 [ISO86] and can be used in any DTD. They need not be 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' instances 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



4: D-STREAT - Dual Model

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. Interpretation of SGML attributes is only possible for a limited set of attribute types being part of [ISO86], such as `ID`, `IDREF`.

Type-to-Class Mapping. Figure 4 is a fragment of Figure 3: The boxes next to the objects indicate from which types the objects inherit their properties 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.

Illustration. Extended functionality, as compared to other 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 following code fragment is part of a sequence of operations generating an instance of the DTD in Figure 1.

```
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
```

5: Example

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

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.

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. – Versioning features are currently integrated into D-STREAT.

Coupling with SGML Parser. To insert documents as a whole into the system, the SGML parser ASP ("Amsterdam Parser") [WaE87] is coupled with D-STREAT. 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 describing aggregation 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. We cannot yet answer the question whether current OODBMS technology is able to handle large document bases. Namely, VODAK is a prototype with considerable leeway for optimization. Documents stored in OODBMSs according to their tree structure naturally occupy more disk space than as a file: With “extreme” documents consisting of as much markup as possible the factor in VODAK is about a hundred. However, with conventional OODBMSs semantic control over the objects is not part of the database. This would lead to a substantially reduced disk-space consumption, but there would be no basis for semantics–based query optimization [AbF93]. Setting up the entire document trees in the database is rather slow. It is comparable to the time interval for establishing the database structures in the loose coupling between INQUERY and an OODBMS described in [CST92]. The duration for large documents is in the hour range. – We will assess query evaluation with our database application after integrating VODAK query-optimization features.

5 Related Work

We are not aware of OODBMS applications for structured document storage. However, research in various areas has contributed useful findings.

Document Storage. In [CrS87, ZTS91, CST92, Bar+93] either document structures for database storage or database applications for document handling are described. As opposed to our approach, only special document types are dealt with. Three possible kinds of textual document storage in nested relational database systems are discussed in [ZTS91]. The article’s objective is to compare the efficiency of the different techniques presented. The approaches are static, i.e. the document type is hardcoded in the schema. The Gold Mailer [Bar+93], an integrated system for sending and receiving messages, contains a so-called index engine for archiving received messages. Again, the document structure is fixed. In queries structure-orientation can be combined with content-orientation. In this context, however, content-based retrieval is pattern matching. A loose coupling of the text-retrieval system INQUERY with an OODBMS is described in [CST92]. Likewise, there is a rigid set of document-component types. On the other hand, INQUERY’s retrieval functionality is an asset.

The objective of the Rufus System [Sho+93] is to provide database support for semi-structured data. For them, textual documents are just an arbitrary example of semi-structured data next to, say, computer programs or images. Hence, document-specific support (e.g. by using concepts such as SGML) is not envisaged. Rufus does not modify the file objects themselves, because the developers want to carry on using applications developed against the data's original format. We argue that the negative valuation of approaches that transform documents' formats to a database format is too undifferentiated. Namely, storage of documents in the database is done to extend functionality. The decision whether document storage in databases is advantageous is manifold (cf. Section 1). With Rufus the database contains descriptive information about file-system objects. It is only the retrieval functionality that is extended. – With the system presented in [Bur92] it is possible to formulate queries referring to both the structure as well as its content. As opposed to the approaches mentioned before, arbitrary document structures can be dealt with. Again, however, content-based search is mere pattern matching. This functionality can in principle be achieved with our system by integrating a grep-command for the VML-datatype STRING. – It seems that with these approaches documents' modification is not supported.

SGML Databases. Some years ago pioneering work has been done in the field of storing SGML documents in databases. Schouten [Sch89] describes how with his system SGML documents are fragmented and how the fragments are stored in a relational DBMS. The mapping from SGML documents to the relational schema is intricate. It is not part of the system or of an application program. Hence, the system does not capture SGML's semantics. Documents' organization in the database has no DTD-specific aspects: The data structures are generic and are not adapted to individual DTDs. However, avoiding these pitfalls seems to be difficult using a relational system.

Schema Evolution. With main-stream OODBMSs a schema consists of class definitions[5] containing the definitions of instances' properties and methods and subclass-relations between classes. Correspondingly, on a more formal level an object is specified by its signature and a set of axioms [SSC92]. Axioms being first-order-predicate-logic formulae correspond to that conventional notion of 'schema' without a metaclass mechanism as in VML. From that perspective, operations within D-STREAT such as creating a new class are schema-change operations. With the VML conception,

5. The distinction between object types and classes in VML is rather an exception, as compared to other OODBMSs.

metaclass definitions may be part of the schema. This corresponds to second-order-predicate logic. In this case, creating new application classes is not schema evolution.

6 Conclusions

The design of a database application for structured documents (D-STREAT) has been elaborated. D-STREAT is currently being refined. The documents are fragmented for storage according to their SGML-document-type definition. D-STREAT's functionality is different compared to systems that use databases only to store description information on the documents or that carry out a generic fragmentation independent of the document-type-specific structure. The OODBMS VODAK offers the advantages of differentiating between classes and types, of metaclasses to model semantic relationships between classes and of generating instances of metaclasses at runtime.

On the other hand, IRSs have features that D-STREAT currently does not have. The idea to develop a coupling between an IRS and D-STREAT is nearby. On the one hand, having the functionality of both IRSs and database applications for document storage is envisioned. Furthermore, new functionality results from the combination of IRSs with DBMSs. For instance, it would be possible to formulate queries referring both documents' content as well as the one of databases. Query optimization using knowledge on operations' costs is currently being integrated into VODAK [AbF93]. Because costs of IRSs' operations are principally known the next goal would be an integrated query optimization taking into account both the database methods and the IRS's operations. An interesting topic in this context would be to exactly identify situations in which redundant document storage, that is storage both as one large file object and as fragments, is advantageous.

Another issue is database support for generic structures. In this case, tree structures are of interest. In [Sub+93] it has basically been pointed out that there is the potential for query optimization. We are, however, not aware of a realization of query optimization for tree structures.

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 the worst mistakes.

References

AbF93 K. Aberer, G. Fischer, "Object-Oriented Query Processing: The Impact of Methods on Language, Architecture and Optimization", *Arbeitspapiere der GMD No. 763*, St. Augustin, 1993

ABH94 K. Aberer, K. Böhm, C. Hüser, “The Prospects of Publishing Using Advanced Database Concepts”, to appear in *Proceedings of Electronic Publishing 94*

AkQ92 E. Akpotsui, V. Quint, “Type Transformation in Structured Editing Systems”, in C. Vanoirbeek, G. Coray (eds.), *Proceedings of Electronic Publishing 92*, Lausanne, Switzerland, Cambridge University Press, pp. 27-42

Bar+93 D. Barbará et al., “The Gold Mailer”, in *Proceedings of the Ninth International Conference on Data Engineering*, Vienna, Austria, IEEE Computer Society Press, pp. 92-99

BIP92 N. Belkin, P. Ingwersen, A.M. Pejtersen (eds.), *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1992, ACM Press

Bur92 F.J. Burkowski, “Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text”, in [BIP92], pp. 112-125

CrS87 W.B. Croft, D.W. Stemple, “Supporting Office Document Architectures with Constrained Types”, in *Proceedings ACM SIGMOD 1987*, San Francisco, May 27-29, pp. 504-509

CST92 W.B. Croft, L.A. Smith, H.R. Turtle, “A Loosely-Coupled Integration of a Text Retrieval System and an Object-Oriented Database System”, in [BIP92], pp. 223-232

ISO86 Information Processing – Text and Office Systems – Standardized Generalized Markup Language (SGML), ISO 8879-1986 (E), *International Organization for Standardization*, 1986

ISO89 Information Processing – Text and Office Systems – Office Document Architecture (ODA) and Interchange Format, Part 2, Document Structures, 1989

Her90 E. van Herwijnen, Practical SGML, Kluwer Academic Publishers, 1990

Hal94 M. Halper et al., “Integrating a Part Relationship into an Open OODB System Using Metaclasses”, submitted to ADB-94

KAN93 W. Klas, K. Aberer, E.J. Neuhold, “Object-Oriented Modeling for Hypermedia Systems using The VODAK Modeling Language (VML)” in *Object-Oriented Database Management Systems*, NATO ASI Series, Springer Verlag Berlin Heidelberg, August 1993

Kla+93 W. Klas et al., “VML - The VODAK Model Language Version 3.1” *Technical Report, GMD-IPSI*, July 1993

Sch89 H. Schouten, “SGML*CASE The Storage of Documents in Databases”, Vol. 4, No. 1, SGML Users’ Group Bulletin, 1989

Sho+93 K. Shoens et al. “The Rufus System: Information Organization for Semi-Structured Data”, in R. Agrawal, S. Baker, D. Bell (eds.), *Proceedings of the International Conference on Very Large Data Bases 1993*, Dublin, Ireland, VLDB Endowment, pp. 97-107

SSC92 A. Sernadas, C. Sernadas, J.F. Costa, “Object Specification Logic”, INESC Research Report 1992, to appear in *Journal on Logic and Computer Science*

Sub+93 B. Subramanian et al., “Ordered Types in the AQUA Data Model”, *Proceedings of the Fourth International Workshop on Database Programming Languages 1993* (DBPL)

WaE87 J. Warmer, 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

ZTS91 J. Zobel, J.A. Thom, R. Sacks-Davis, “Efficiency of Nested Relational Document Database Systems”, in G.M. Lohmann, A. Sernadas, R. Camps (eds.), *Proceedings of the International Conference on Very Large Data Bases 1991*, Barcelona, Spain, VLDB Endowment, pp. 91-102