View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoscience - École polytechnique fédérale de Lausanne

appeared in: Proc. of the 3rd International Conference on Information and Knowledge Management (CIKM) '94, Maryland, MD, 1994.1

Storing HyTime Documents in an Object-Oriented Database

Klemens Böhm, Karl Aberer
GMD-IPSI
Dolivostraße 15, 64293 Darmstadt
Germany
{kboehm, aberer}@darmstadt.gmd.de

Abstract

An open hypermedia-document storage system has to meet requirements that are not satisfied by existing systems: it has to support non-generic hypermedia document types[1], i.e. document types enriched with application-specific semantics. It has to provide hypermedia-document access methods. Finally, it has to allow the exchange of hypermedia documents with other systems. On a technical level, an object-oriented database-management system, on a logical level, a well established ISO standard, namely HyTime, is used to satisfy the requirements mentioned above. By means of the example of documents incorporating hypertext structures we discuss the impact of taking such an approach on representation and processing within the database system.

1 Introduction

In the recent past the proliferation of the hypertext paradigm for representation of information has been facilitated by technological advances. The main difference between conventional documents and hypertext documents is that in the first case the document structure as perceived by the reader is linear. In the second case, there is a graph structure. To apply the hypertext paradigm *hyperengines* have been developed. Hyperengines administer the hypertext-specific structures such as nodes, links and anchors[2] which will be referred to as *hyperobjects*. They contain the realization of generic operations. An example for a node operation is the calculation of the transitive closure, i.e. the identification of all nodes in a document that can be reached by link traversal from that node. Hyperengines differ in the internal representation of the hyperobjects. Furthermore, operations reflecting the hyperobjects' semantics may be more or less sophisticated. Usually systems for hypertext-document handling consist of three layers, a *storage layer*, an *application layer* and a *presentation layer* [DeS86]: hyperengines are the middle layer. With some hyperengines, the storage layer is made up with databases [ScS90, MaS92]. With other ones, this is not the case [SSS93].

[1] In our terminology, with hypermedia documents the content of the nodes may be multimedia data as opposed to hypertext documents.

[2] The edges of such a kind of graph structure are called links. Links do not necessarily link nodes in their entirety, but also structures within nodes as, say, several words or sentences. The link ends are called anchors [Con87].

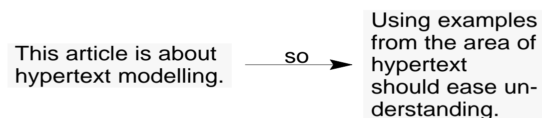

Figure 1: Sample Structure in an Argumentation Space

In hypertext-documents of different types the hyperobjects have special semantics in addition to the canonical features. In [SHT89] four spaces corresponding to the different design activities within an authoring process are identified: the *content space*, the *planning space*, the *argumentation space* and the *rhetorical space*. These spaces are part of SEPIA, a cooperative hypermedia authoring environment developed at our institute [Str+92]. An argumentation space inter alia contains facts ('datum') and assertions ('claim'). There are not only different node types, but also different link types: a so-link is a directed binary link from a datum to a claim. Figure 1 contains an example of a so-link. Other link types in the argumentation space are the contributes_to-link and the contradicts-link. A contradicts-link links two nodes with discrepant content. In the other spaces the hyperobject's semantics likewise is special. In the sequel we will refer to an argumentation-space structure, i.e. a hypertext structure whose components are claims, so-links etc. as an *argumentation-space document*.

In existing systems hyperobjects' document-type-specific semantics tends to be hardcoded in the presentation layer. Hence, exchanging hyperdocuments of non-generic types, e.g. argumentation-space documents between hyperengines or applying a hyperengine in different contexts is not yet conceivable. We for our part envisage a hyperengine supporting hyperobjects with partly special semantics based on a database system. We want to comply to a format for hyperdocuments satisfying the following basic requirements: non-genericity, orientation towards hypermedia document storage and processing, acknowledgement as an (international) standard. Hence, we have chosen SGML/HyTime.

The problems we approach and solutions we give are fairly orthogonal to the particular format chosen.

With SGML ('Standard Generalized Markup Language') [ISO86, Her94] document types can be defined. In essence, *SGML document-type definitions (DTDs)* are attributed grammars specifying the document structure. However, nothing is said about the semantics of document components, which are called *elements* in the SGML context. The HyTime Standard ('Hypermedia/Time-based Document Structuring Language') [ISO92, NKN91] basically is a list of SGML element-type definitions for, say, links or presentation schedules. These element types are referred to as *architectural forms*. Their semantics is fixed by the standard. - In this article we focus on the basic link features the HyTime standard provides. This facilitates a comparison of our concepts to conventional approaches to hyperdocument storage. We are, however, not aware of any related work on this topic dealing with different hypertext-document types.

In this article we describe the database application framework for HyTime-document storage we are currently working on. We concentrate on the following aspects.

1. In [ISO92, Koe+93] it has been mentioned that HyTime processing can be accelerated by means of an internal representation. Our objective is to develop hypermedia-specific index structures to speed up access operations. We will explain that having more than one internal representation of HyTime-architectural forms' instances to choose from may be advantageous.

2. We do not restrict ourselves to a set of fixed hypertext structures: dynamic modifications of SGML/HyTime documents shall be doable. With our approach both the collection of documents and the set of document types can be modified at runtime.

3. Operations are part of the database application: In other words, the database has the semantic control over document components.

The platform on which realization will be based is an object-oriented database-management system (OODBMS) - the OODBMS VODAK developed at our institute [Kla+93, KAN93]. By using a DBMS database features such as concurrency control or querying capabilities are available. Even though we limit ourselves to the description of the HyTime hyperlink features other facets of hypermedia documents reflected in HyTime, like spatial and temporal relationships, can be approached in the same way.

The structure of this article is the following: the next section is an overview of the HyTime conception together with examples from the hypertext area. In Section 4, our database-based approach to HyTime-document handling is discussed, and it is described how to realize the link features. We review and classify related work in Section 5. Section 6 is a brief summary, and we identify further research objectives.

2 HyTime

SGML. Structured documents may be seen as trees whose edges indicate how components are contained in each other. The tree corresponding to this article is depicted in Figure 2. This hierarchical structure is commonly referred to as the *logical document structure*. The nodes are the elements, the list of a node's children the *content of the element*. Internal nodes are *nonterminal elements*. All elements have a type, e.g. `section` or `paragraph`. With SGML the logical structure of documents of a certain type can be defined. In essence the *content model of an element type* is a regular expression specifying how the content of an element of that type may look like. An element-type name is also called *generic identifier (GI)*. The element-type definitions in a DTD may be completed by the definition of attributes.

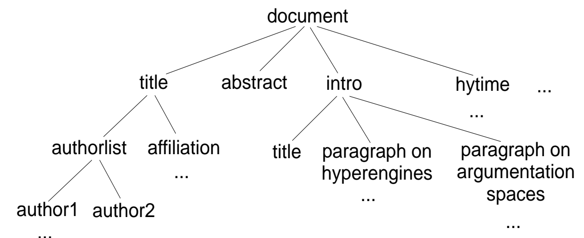

Figure 2: Tree Structure Corresponding to this Document

```
<!ELEMENT asdoc       (node|so|contra|contrib)*>
-- 'asdoc' short for 'argumentation-space
   document', contra' is GI of 'contradicts-link',
    'contrib' is GI of 'contributes_to-link'
    (def.s omitted) --
<!ELEMENT node  CDATA>
<!ATTLIST node  id      ID       #REQUIRED
                type    (position|claim|datum)
                        #REQUIRED>
<!ELEMENT so    EMPTY>
<!ATTLIST so    claim   IDREF         #REQUIRED
                datum   IDREF         #REQUIRED>
...
```

Figure 3: Fragment of an SGML Document Type Definition for Argumentation-Space Documents

Figure 3 contains a possible DTD for argumentation-space documents. Lines starting with '`<!ELEMENT`' introduce an element type together with its content model: instances of `asdoc` contain a list whose elements are either instances of element type `node`, `so` etc. `CDATA` is a terminal element type more or less comparable to the data type `STRING`. '`<!ATTLIST`' indicates the beginning of attribute definitions. For instance, elements of type `node` have an attribute of type `ID` and an attribute `type` of type (`position|claim|datum`). Attributes of type `ID` are unique identifiers of the element they belong to. An attribute of type `IDREF` is an ID reference, one of type `IDREFS` a list of ID references. `EMPTY` and `#REQUIRED` are SGML keywords being of minor importance in this context. Comments are enclosed in double hyphens.

```
<asdoc> <node id="n1" type=claim> Using examples
from the area of hypertext should ease
understanding.</node> <node id="n2" type=datum>
This article is about hypertext modeling.</node>
<so datum="n2" claim="n1"> </asdoc>
```

Figure 4: SGML Example - Fragment of a Document Corresponding to the DTD in the Previous Figure.

It is important to notice that with SGML it is basically the logical document structure that is described by means of an attributed grammar. One of the few provisions on the semantic level that is part of the standard is the semantics of attributes of type `ID` and `IDREF`. Figure 4 contains a fragment of a document in conformance with the DTD from Figure 3. In the sequel, we refer to documents conformant to a DTD as *instances of the DTD*. The document corresponds to the structure depicted in Figure 1. The example also illustrates that the use of SGML is not limited to conventional text documents, but instead can also be used for hypermedia documents.

Basic Concepts of HyTime. The processing semantics of conventional document-element types such as `section`, `chapter` or `paragraph` is independent of the element type. For instance, for all of these element types there might be a method to display the textual content on the screen. With hypermedia documents the element types' semantics is more differentiated and not necessarily obvious: suppose that in an argumentation space one may navigate through a so-link only from the datum to the claim, but not in the opposite direction. Constraints of that kind make up the *browsing semantics*. The anchors of a contradicts-link, to give another example, are equivalent: the browsing semantics would therefore be different from the browsing semantics of instances of `so-link`. To reflect the browsing semantics of link elements in an SGML document one might introduce an attribute bearing that information. The interpretation of attribute values of that kind, however, is not standardized. If an application were to process instances of a DTD with such an attribute it would have to be adapted to the DTD by hand. Suming up, SGML DTDs are not well-suited as exchange formats for hypermedia documents. The HyTime standard has been designed to overcome these problems: it essentially is a list of SGML element-type definitions called *element-type forms* or *architectural forms*. The special feature of HyTime is that architectural forms' semantics is fixed by the standard. The semantics, e.g. the attributes' meaning, is partly described in natural language, in part it follows from the comment[3] within the forms' definitions. Element types in SGML/HyTime DTDs may be specializations of HyTime element-type forms. The meaning of 'specialization' in this context is twofold:

- The specialization of an element-type form may contain additional attributes. On the other hand, it need not contain all attributes of the element-type form. Consider the case of hypertext documents being so small that there is no need to navigate through them because they can be viewed in their entirety. In that case, the attributes `extra` and `intra` can be omitted.
- The ranges of the content and of the element-type form's attributes may be subsets of the original ranges in the HyTime standard.

A type definition for links, to continue the example, should contain the information in which direction navigation is possible. The relevant element-type form from the standard, whose name is `ilink`, has attributes to bear that kind of information. The names of these attributes are `extra` and `intra`. `ilink` ('independent link') is the element-type form of which link element-type definitions should be specializations. In Figure 4, we have shown how such a structure can be modelled using SGML only. We are now in the position to illustrate how the same structure can be represented using the HyTime element-type form `ilink`. Figure 5 contains the relevant portion of the DTD. The fact that `so` is a specialization of `ilink` is indicated by setting the attribute `HyTime` to `ilink`. Figure 6 is the document fragment corresponding to Figure 1 as an instance of the HyTime-DTD. The attributes displayed in the figure (except for `reftype` which will be explained in the following paragraph) are inherited from the element-type form `ilink`: `anchrole` introduces a label for each anchor, in this case `DATUM` and `CLAIM`. `reftype` specifies the types of the elements referenced by another attribute: `"linkends anchors #SEQ"` means that a value of `linkends` must be conformant to the content model of `anchors`, i.e. an admissible value of the `linkends`-attribute is the ID of a `datum`-element, followed by the ID of a `claim`-element. The attribute `extra` is to capture the browsing semantics. Here, we will not explain the meaning of the attribute value. In this context, it is sufficient to know that its meaning is specified by the standard.

```
<asdoc> <node id="n1" type=claim>Using examples
from the area of hypertext should ease
understanding.</node> <node id="n2"
type=datum>This article is about hypertext
modeling.</node> <so linkends="n2 n1" ...
extra="A E" ...> </asdoc>
```

Figure 6: HyTime Example - Fragment of a HyTime Document Corresponding to the DTD in the Previous Figure.

Additional Terminology. As opposed to element-type forms containing both content's and attributes' definitions an *attribute-list form* is a list of attribute definitions only. Again, the standardization of the attributes' semantics is of importance. An architectural form is either an element-type form or an attribute-list form. The attribute `reftype` mentioned previously is part of a HyTime attribute-list form. An attribute of this type may be included in element-type definitions in HyTime documents. Again, the attribute values' meaning, e.g. how to interpret the value `"linkends anchors #SEQ"` is described in the standard. The HyTime standard consists of several modules. Each of them contains a list of architectural forms. The prologue of a HyTime document contains so-called *support declarations* stating which features, e.g. architectural forms, need be supported.

Note that attributes whose value is already fixed in the DTD can be omitted from document instances, such as attributes `HyTime` and `anchrole` in Figure 6.

3 Requirements on HyTime Document Storage

SGML. In [ABH94, BAH93] our approach to SGML document storage has been described. There the following requirements have identified inter alia:

- It shall be possible to alter parts of the documents without locking off the whole document for other authors. Namely, we envision our database application to be a platform also for authoring systems. Multi-authoring may be facilitated by concurrency control.

[3] The name 'comment' is misleading, because in HyTime comment may have binding forces. Comments in HyTime element-type definitions can further restrict the range of attributes or of the content. Hence, an SGML document whose DTD contains a HyTime element-type form might be a correct SGML document, but not a valid HyTime document.

```
<!ELEMENT asdoc    (position|claim|datum|so|contra|contrib)*>
<!ELEMENT claim    CDATA>
<!ATTLIST claim    id        ID>
<!ELEMENT datum    CDATA>
<!ATTLIST datum    id        ID>
<!ELEMENT so       EMPTY>
<!ATTLIST so       HyTime    NAME   #FIXED   ilink
                   anchrole  NAMES  #FIXED   "DATUM CLAIM"
                   linkends  IDREFS #REQUIRED
                   reftype   CDATA  #FIXED   "linkends anchors #SEQ"
                   extra     ...>
<!ELEMENT anchors  (datum, claim)>
```

Figure 5: Example - Fragment of a HyTime Document Type Definition

- The DBMS must have semantic control over versions of document elements and the generation of new version objects.
- The usage of multimedia types shall be supported by the underlying system [Rak+93].
- It is not worthwhile to be restricted to a fixed set of document types. On the contrary, it shall be possible to administer arbitrary DTDs. Furthermore, it occurs quite frequently that document types change over time. DTD handling should be as simple as possible.

In [ABH94] we have claimed that these requirements can best be met using an OODBMS. If document-file objects were left intact, full database functionality would not be achieved. With a generic document fragmentation differences between document types could not be reflected.

HyTime. The requirements for SGML document storage also hold true for HyTime-document storage. Another additional requirement for HyTime documents is that the semantics of HyTime document components, which is fixed by the standard, should be reflected in the database. The database must have semantic control over HyTime objects as a prerequisite for our approach to query optimization [AbF93] and for semantic concurrency control [Mut93]. Because with HyTime operationality is extensive as compared to SGML it is advantageous to put some effort into an efficient realization of the HyTime application-independent processing.

4 A VODAK Application Framework for HyTime Document Storage

The objective of this chapter is to introduce the approach to HyTime-document storage we are currently pursuing. Basic concepts of the VODAK Modeling Language (VML) are reviewed in the next paragraph. More detailed information can be found in [KAN93].

Principles of the VODAK Modeling Language. Objects have *properties* and *methods*. This is mentioned to explain the difference between the usage of 'property' and 'attribute' in this article: attributes are SGML attributes; properties are the variable-like containers for the database objects' data content. The *type* of an object is its property- and method definitions. A *class* is a set of objects of the same type. Objects of the same type need not necessarily belong to the same class. A *metaclass* is a class whose instances are themselves classes. Symmetrically, we refer to the instances of the instances of a metaclass as *metainstances*. The definition of a class encloses the definition of its instances' type. This type is called the *insttype of the class*. A metaclass definition may enclose both the definition of its instances' type and of its metainstances' type. This second type is the *instinsttype of the metaclass*. An object has properties and methods that are defined as part of its class's insttype and its metaclass's instinsttype. Just as classes contain objects with common characteristics, metaclasses are in use to model common characteristics of classes, e.g. semantic relations between classes such as aggregation or specialization. In the context of role specialization, for example, a real-world object has several aspects [SeE90]. For instance, an object might have the generalization aspect 'person' and the specialization aspect 'patient'. In the modeling there are two classes PERSON and PATIENT. Two database objects correspond to each real-world object of that kind: an instance of PERSON and one of PATIENT. The instances of PERSON have person-specific properties and methods, the ones of PATIENT patient-specific ones. Class PATIENT is a *role-specialization class*, its instances are *(role-)specialization instances*. Analogously, PERSON is a *generalization class*, its instances are *generalization instances*. There is a metaclass whose instances are role-specialization classes and whose metainstances are role-specialization instances. The insttype and instinsttype of this metaclass contain the property- and method definitions that are necessary to administer a role-specialization relationship on the class- and on the instance-level, respectively. If the same semantic relationship occurs between other classes, e.g. CAR and SERVICE_VEHICLE, one can fall back upon that metaclass, and the relationship need not be modeled anew.

SGML Layer. Our VML schema for SGML/HyTime documents consists of several *layers*. Layers stand for the different levels of specialization. A document element being instance of a HyTime element-type form has two aspects: the SGML aspect and the HyTime aspect.[4] Within the SGML layer there is a corresponding class for each nonterminal element type. We call these classes *SGML element-type classes*. In Figure 7 there are element-type classes FOOTNOTE, SECTION, CHAPTER, SO-LINK - based on the assumption that the DTD contains element-type definitions `footnote`, `section`, `chapter`, `so-link`. In Figure 7, classes are represented as ellipses, "normal" objects are just dots.

[4] Actually, SGML elements already have two aspects in our modeling, as explained in the previous paragraph, but in this article we abstract from this facet of the modeling.

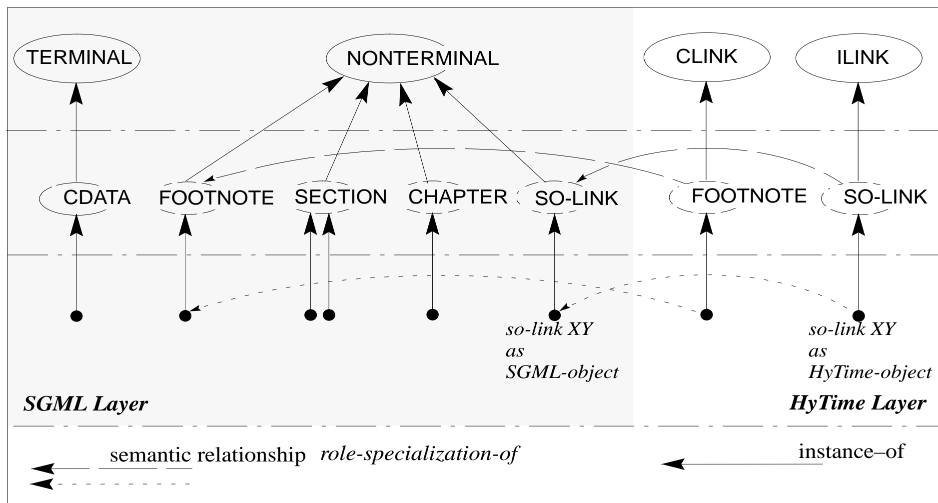

Figure 7: Overview of the Modelling

The plain arrow connects an object with its class. The dashed arrows are from a role-specialization class to the corresponding generalization class or from a specialization instance to the generalization instance. One requirement is that DTDs may be inserted dynamically into the document base as they are not known ahead of time. Hence, element-type classes may be generated dynamically. Classes that are generated dynamically are represented by a dashed-line ellipse. Dynamic generation of element-type classes is possible because the instances of element-type classes are of the same type, independent from their particular element-type class. This in turn is possible because the processing semantics of nonterminal SGML-element types is generic, leaving aside the HyTime context. Terminal element types such as `CDATA` are DTD-independent. The corresponding classes to comprise, say, `CDATA` elements are part of the schema, e.g. class `CDATA` in Figure 7. The instances of different terminal element-type classes, on the other hand, are not of the same type: `CDATA` elements, to give an example, have a method to display textual content that does not necessarily make sense for other terminal element types, especially if datatypes for continuous media data are involved.

Interpreting SGML attributes of a freely-defined type is not part of the system. This is different for attribute types being part of ISO 8879-1986 such as the ones of type `ID` or `IDREF`. Experience shows that the non-hierarchical document structure induced by these attributes is of minor importance with conventional textual documents. This is the reason why the `ID/IDREF`-semantics will be used to illustrate how HyTime features shall be realized.

HyTime Layer. A HyTime document element has both SGML semantics and HyTime semantics: the SGML semantics, to give examples, is reflected by operations to navigate through the parse tree or to verify whether an element's content conforms to its content model. Operations reflecting HyTime semantics are of course different for individual architectural forms. In the database there are two objects corresponding to a HyTime document element, an SGML object and a HyTime object. The HyTime object is role-specialization instance of the SGML object. We call the role-specialization classes *HyTime element-type classes*. An overview of the modeling is in Figure 7. As classes in the SGML layer are created dynamically, this must also be the case for their role-specialization classes, i.e. classes in the HyTime layer. In principle, there is a metaclass for each HyTime element-type form in the HyTime layer. E.g. ILINK in Figure 7 corresponds to **`ilink`**, CLINK to **`clink`**.[5] The definition of HyTime-specific operations is part of these HyTime metaclasses' instinsttype. Just as SGML element-type classes are essentially container for elements of the same type, HyTime element-type classes are container for the HyTime role-specialization instances: they contain the specializations of the SGML objects of one element type. Classes and objects in the HyTime layer are generated only when the relevant support declarations are part of the HyTime document's prologue.

It has been mentioned previously that role-specialization classes are instances of a metaclass for role specialization. This metaclass provides the properties and methods to administer role-specialization relationships for their instances and metainstances. On the other hand, however, metaclasses in the HyTime layer contain the type definition of the HyTime objects, i.e. HyTime-specific properties and methods. Hence, the type definitions in HyTime metaclasses must provide both properties and methods for both facets.

HyTime Index Structure. Document exchange formats need not necessarily be identical with the internal format. Namely, system peculiarities must be taken into account to find the optimal format. Instead of transforming the document into a format different from the one for SGML documents we suggest a more differentiated view: transforming documents into an internal format makes incremental changes more difficult. In other words, a view on this internal format would have to exist. With 'internal format' we refer to a format different from the one for SGML documents. By introducing such an internal format, SGML functionality would have to be adapted. With our current approach to SGML-document storage SGML opera-

[5] ilink and clink are the element-type forms being part of the HyTime Hyperlink module. ilink has already been explained to some degree. Here we just mention that clink ("contextual link") is also a link structure, but one simpler than ilink.

tions are integrated into the modelling in a natural way. To solve this problem we are developing HyTime-specific index structures: on the one hand the SGML format is preserved, on the other hand HyTime functionality is accelerated. Just as conventional index structures are adapted to changes of the data, modification of the HyTime-index structures likewise is triggered by update operations on the document.

Of course it would be possible that the SGML format, i.e. the properties of the objects in the SGML layer, could serve as a basis for the processing of the HyTime objects. Without the index structures the HyTime application-independent processing would have to rely on the SGML format, i.e. the properties of the SGML objects that at the same time are generalization instances of HyTime objects. For instance, consider a method for the element-type form `ilink` that, given an anchor-role name, returns the corresponding anchor. Based on the SGML format, its execution would be as follows: first, the SGML object's property corresponding to the attribute `linkends` would be accessed. By parsing the property value the character sequence corresponding to the ID of the anchor object can be determined. Next, the SGML objects are searched for the one with that ID. It is returned when found. However, in order to keep read-access operations cheap, HyTime attributes and content are not only stored as properties of the SGML objects, but also a second time as properties of the HyTime objects. For this redundant storage, another format is used. With our system incremental updates shall be possible. Whenever an SGML object's property is modified the corresponding properties of the HyTime object are also updated. This conception corresponds to general indexing mechanisms in databases: read operations are accelerated by means of redundant storage. When the data is altered the index structure is updated correspondingly. In the sequel, we exemplify these notions by means of the HyTime link features. For the moment, we limit ourselves to a basic set of features. We concede that with this restriction full HyTime expressiveness is not yet achieved. However, to model a basic set of hypertext structures, this is sufficient. Besides that, it is adequate to deal with the document structures of SEPIA.

Modeling HyTime Link Features. The instances of the HyTime element-type form `ilink` have an attribute `linkends` of type `IDREFS`. An SGML object representing such an element has a property containing the attribute value. The specialization instances, i.e. the HyTime objects, have a corresponding property whose type is a list of OIDs[6]: each OID corresponds to an ID reference. It is important to notice that while OIDs identify database objects, SGML attributes of type `ID` or `IDREF` identify SGML document components, i.e. entities on another logical level. When the `linkends` attribute value of the corresponding SGML object is altered that list is updated. I.e. the OIDs of the anchor objects are identified and entered into the list. Thus, searching for the link anchors is not necessary within read operations, but instead direct access is possible. By explaining that SGML IDs can be replaced by VML OIDs we rather want the different conceptual levels to become evident: SGML attributes (e.g. the `linkends` attribute) are on a level that we call the logical one, the VML representation (such as the list of OIDs) is on a level that we refer to as the physical one. With HyTime attributes from other HyTime modules the difference between these conceptual levels is sometimes hardly perceivable. Sometimes the mapping between the levels is less definite and depends to a stronger degree on the modeling language's constructs.

HyTime Attribute-List Forms. In this paragraph we briefly describe the way HyTime attribute-list forms are dealt with. The attribute `reftype` is used as an example. This particular HyTime attribute is described because it is - just as `ilink` - indispensable to model hypertext structures in HyTime. The attribute `reftype` is not an attribute of individual elements. It rather is a supplement of element-type definitions having an attribute of type `IDREF`. Thus, the `reftype`-semantics is element-type-specific: there is no need for a role-specialization object for each instance of the SGML element-type class. Instead one single role-specialization object of the SGML element-type class as a whole is sufficient to capture the `reftype`-semantics. The `reftype`-semantics is basically reflected in a method checking whether the instance of the `IDREF` attribute conforms to the `reftype`-value. The method is triggered whenever the corresponding `IDREF` attribute is updated.

"SGML Correctness" vs. "HyTime Correctness". An SGML document that is correct need not necessarily be conformant to the HyTime standard. For instance, leaving aside the compulsory HyTime comment, the `ilink` attribute `anchrole` is a list of names, and the attribute `linkends` is a list of SGML IDs. The fact that the number of names must equal the number of IDs is stated by the HyTime comment. An element with different numbers of anchor-role names and SGML IDs may be correct according to the SGML DTD. It is, however, not correct according to the HyTime standard. In that sense, we provide for flagging HyTime objects as correct or not correct. We see no other way to ensure that methods relying on the internal format of the HyTime attributes work correctly. Checking conformance of HyTime element-type forms' instances to the restrictions in the HyTime comment is part of the HyTime application-independent processing.

Further Refinements of the HyTime Layer. Up to this point it has been suggested that there is one metaclass for every HyTime element-type form. However, it is conceivable to have more than one metaclass corresponding to an element-type form. One reason might be to improve performance, another one to extend functionality. A special kind of independent links, to give an example, are binary links, i.e. links with two anchors. First, assume that the number of anchors is bigger than two. In that case, administering the anchors in the HyTime object by using a list is adequate. If, however, there are two anchors using a list as the relevant property type of the HyTime object is generally slower than just having two properties for each anchor. On the other hand, there may be methods that are special for a particular link type. For instance, consider a method getOtherAnchor whose parameter is one anchor and whose return value is the other one. It makes sense for binary links only. Selecting the right metaclass, i.e. the appropriate internal format, can be accomplished by the system.

The HyTime semantics is not only reflected in methods of HyTime architectural forms' instances. There are no element-type forms for nodes, i.e. elements referenced by `ilink`-instances, to give an example. Nodes may be elements of arbitrary types. On the other hand, nodes have processing semantics, too. An example of an operation capturing

[6] The type of the logical units' identifier in the VODAK OODBMS is referred to with OID ("Object Identifier").

the semantics would be methods to calculate the transitive closure, another example would be a method identifying a node's adjoining links, possibly only links of a certain type.[7] In analogy to role-specialization objects in the HyTime layer bearing the semantics of HyTime element-type forms there are role-specialization objects with node semantics for all elements in documents with the `ilink` support declaration. In other words, in addition to metaclasses such as ILINK there are metaclasses such as INVERSE_ILINK. Again, the metainstances inherit the node semantics from the metaclass.

5 Related Work

Research in various areas has contributed useful findings. In this section, they are brought in relation to our work.

Structured Documents. The relationship between structured documents and hypertext structures has been investigated before, e.g. in [QuV92]. With structured documents the structure of the document components is hierarchical. This need not be the case with hypertext structures. The editor Grif described in [QuV92] can handle both hierarchical and non-hierarchical document structures according to arbitrary DTDs. A proprietary format is used, but on the conceptual level that DTD corresponds more or less to SGML DTDs. While we are working on a storage system, Grif is an editor: because with an editor documents are only edited and not processed, the question whether HyTime-like concepts, i.e. certain document components' semantics, have been taken into account would not make sense.

Hyperengines. There exists a broad spectrum of data models for hypertext. Based on these, various hyperengines have been built. [CaG88, ScS90, MaS92, SSS93] is merely a selection. The data models reflect the generic, document-type independent semantics of document components from the hyperengine developers' point of view. With these hyperengines it is tacitly assumed that the document-type-specific semantics is part of the presentation layer, i.e. the application on top of the hyperengine. On the other hand, with the HyTime approach not the entire document-type-specific semantics needs to be realized anew for every document type by reimplementing the presentation layer. Namely, in parts it is already contained in the element-type definitions - the ones being specializations of HyTime architectural forms. One objective of a large part of these articles was a refinement of the generic semantics, which is reflected in the operations. In [MaS92], for example, several kinds of delete-operations for nodes are described: in case of "reckless delete" a node is deleted together with all links referencing it; in case of "content-based delete" a node is deleted, and links that previously referenced it now contain an invalid reference; and in case of "considerate delete" a node is deleted after verifying that there are no links referencing it. Operations such as these could be part of the HyTime application-independent processing for nodes. I.e. metainstances of INVERSE_ILINK could be deleted in three different ways. The objective of our work, however, is not to polish the operationality reflecting the semantics of hypermedia document components. Instead we want to investigate how to integrate it into the SGML/HyTime context.

Extensible Hypertext Systems. Taking into account the variety of existing hyperengines in [WiL92] it is observed that no internal structure succeeded to be a standard or quasi-standard. The authors follow that further experimentation with hyperengines is necessary. Allegedly, this is considerably eased with the system described: in principle a collection of classes is made available from which hyperengines can be constructed. In the article the construction of two existing engines is summarized. In clear contrast to work discussed in the previous paragraph maximal flexibility is announced. The reverse of the medal is that one still has to do implementation work to arrive at a new hyperengine. However, we claim that the quest for the optimal internal structure is not the only core problem. Another important problem is to determine appropriate exchange formats for hypertext-documents. Furthermore, the optimal structure certainly on the one hand depends on the storage layer and on the other hand on the frequency of the different kind of operations. Hence, we claim that there simply is not just one optimal structure. In a way, our approach is between this one and the conventional hyperengines described in the previous paragraph: flexibility is achieved by processing arbitrary HyTime DTDs. On the other hand, nothing needs to be reimplemented.

HyTime. A running system in the HyTime area is described in [Koe+93]: processing of a HyTime document consists of three phases. First, there is an SGML parsing process. Second, the HyTime engine called HyOctane does additional checking and creates internal structures. Third, the document is presented. Our long-term objective is to realize a relevant part of the HyTime semantics as part of a database application. With HyOctane databases are used as mere storage systems. It seems that full database functionality has not been envisaged.

An important question in this context is the one about HyTime's limits. As opposed to MHEG [KrC92, ISO93, Pri93] information concerning the presentation, e.g. whether in the user interface buttons or sliders are to be used, is not part of the document. In other words, a consensus on what a document is has not yet fully emerged. On another level, we see the problem that only a fragment of hypermedia-document components' semantics can be mapped on the database objects' attributes. There is ongoing research concerning documents components' semantics that cannot be expressed with HyTime. In [ZhP92], to give an example, it is described how can be applied to describe the browsing semantics of hyperdocuments. On the other hand, in HyTime there are the attributes `extra` and `intra` to capture the browsing semantics. With these attributes it is merely the last traversal action that can be taken into account to determine the traversal actions allowed. The flexibility of is naturally higher: a sequence of traversal actions can be evaluated to identify those actions. The complexity of the HyTime standard would considerably increase if such information was also given within the document. A question that cannot unmistakably be answered is what part of the semantics, e.g. the browsing semantics, should be part of the document. It is also conceivable to leave this to the application or the "reader's" preferences.

[7] With regard to the nodes' semantics a deeper view is advantageous. Generic operations such as the general calculation of the transitive closure are part of the SGML semantics: this is because to this end only the attributes of type ID/IDREF are needed. With HyTime the functionality may be more differentiated, e.g. the transitive closure for certain link types or anchor types can be calculated: this would be part of the HyTime semantics.

6 Conclusions

We have outlined our approach towards HyTime document storage. The HyTime standard is a 'meta' standard: it can be used to define exchange formats for hypermedia documents. Some of the HyTime link features have been introduced. In the database, there are classes corresponding to element types derived from HyTime architectural forms. They contain the document components with operations reflecting the particular HyTime semantics. The internal structure of HyTime-database objects can be compared to indexing mechanisms within databases, thus facilitating fast access.

We rely on two VODAK conceptions: the distinction between types and classes and the notion of metaclasses. In principle each architectural form corresponds to a metaclass which, in turn, contains the type definition of its instances and metainstances. The instances, "normal" application classes, are containers for document components of the same type - and therefore with the same semantics. These classes can be generated dynamically. Our objective has been not to be restricted to a fixed set of document-type definitions.

The current stage of our work is the implementation phase. However, integrating the HyTime Scheduling Module into our database application imposes yet unsolved problems on the database architecture.

Acknowledgement. We thank Jürgen Wäsch for helpful comments.

References

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

[ABH94] K. Aberer, K. Böhm, and C. Hüser, "The Prospects of Publishing Using Advanced Database Concepts", in *Proceedings of Conference on Electronic Publishing, April 1994*, eds., C. Hüser, W. Möhr, and V. Quint, pp. 469-480, John Wiley & Sons, Ltd., 1994.

[BAH93] K. Böhm, and K. Aberer, "Extending the Scope of Document Handling: the Design of an OODBMS Application Framework for SGML Document Storage", *Arbeitspapiere der GMD No. 811*, Sankt Augustin, 1993.

[CaG88] B. Campbell, and J.M. Goodman, "HAM: a General Purpose Hypertext Abstract Machine", in *Communications of the ACM, July 1988, Vol. 31, No. 7*, pp. 856-861.

[Con87] J. Conklin, "Hypertext: an Introduction and Survey", in *IEEE Computer, 20 (9), Sept. 1987*, pp. 17-41.

[DeS86] N. Delisle, and M. Schwartz, "Neptune: a Hypertext System for CAD Applications", in *Proceedings of the ACM SIGMOD'86 Conference*, Washington DC, U.S.A., May 1986, ACM Press, pp. 132-143.

[Her94] E. van Herwijnen, *Practical SGML* (second edition), *Kluwer Academic Publishers*, 1994.

[Hyp92] *Proceedings of the ACM Conference on Hypertext, Milano, Italy, 1992*, ACM Press.

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

[ISO92] *Information Technology - Hypermedia/Time-based Structuring Language (HyTime)*, ISO/IEC 10744, 1992 (E), International Organization for Standardization, 1992.

[ISO93] *Information Technology - Coded Representation of Multimedia and Hypermedia Information Objects (MHEG)*, ISO/IEC JTC 1/SC 29, International Organization for Standardization, 1993.

[KAN93] W. Klas, K. Aberer, and E. 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.

[Koe+93] J.F. Koegel et al., "HyOctane: a HyTime Engine for an MMIS", in *Proceedings of the ACM Conference on Multimedia 1993*, ACM Press, pp. 129-136.

[KrC92] F. Kretz, and F. Colaitis, "Standardizing Hypermedia Information Objects", in *IEEE Communications Magazine, May 1992*, pp. 60-70.

[MaS92] M. Marmann, and G. Schlageter, "Towards a Better Support for Hypermedia Structuring: the HYDESIGN Model", in *[Hyp92]*, pp. 232-241.

[NKN91] S.R. Newcomb, N.A. Kipp, and V.T. Newcomb, "The "HyTime" Hypermedia/Time-based Document Structuring Language", in *Communications of the ACM, Nov. 1991, Vol. 34, No. 11.*

[Mut93] P. Muth et al., "Semantic Concurrency Control in Object-Oriented Database Systems", in *IEEE Data Engineering 1993*, Vienna, Austria.

[Pri93] R. Price, "MHEG: an Introduction to the Future International Standard for Hypermedia Object Interchange", in *Proceedings of the ACM Conference on Multimedia 1993*, ACM Press, pp. 121-128.

[QuV92] V. Quint, and I. Vatton, "Combining Hypertext and Structured Documents in Grif", in *[Hyp92]*, pp. 23-32.

[Rak+93] T.C. Rakow et al., "Using Object-Oriented Database Systems for Multimedia Applications", in *it + ti - Informationstechnik und Technische Informatik, Themenheft "Multimedia/Hypermedia", Teil 2*, Oldenbourg, Munich, June 1993, pp. 4-17.

[ScS90] H. Schütt, and N.A. Streitz, "Hyper Base: a Hypermedia Engine Based on a Relational Database Management System", in A. Rizk, N.A. Streitz, and J. André (eds.), *Hypertext: Concepts, Systems, and Applications (ECHT'90)*, Cambridge University Press, pp. 95-108.

[SeE90] A. Sernadas, and H.-D. Ehrich, "What is an Object, After All", in R. Meersman, W. Kent, and S. Khosla (eds.), *Object-oriented Databases: Analysis, Design and Construction*, North-Holland, 1991, pp. 39-69.

[SHT89] N.A. Streitz, J. Hannemann, and M. Thüring, "From Ideas and Arguments to Hyperdocuments: Travelling through Activity Spaces", in *Proceedings of the Second ACM Conference on Hypertext (Hypertext'89)*, ACM Press, pp. 343-364.

[SSS93] D.E. Shackelford, J.B. Smith, and F.D. Smith, "The Architecture and Implementation of a Distributed Hypermedia Storage System", in *Proceedings of the Fifth ACM Conference on Hypertext (Hypertext'93)*, Seattle, U.S.A., Nov. 1993, ACM Press, pp. 1-13.

[Str+92] N.A. Streitz et al., "SEPIA: a Cooperative Hypermedia Authoring Environment", in *[Hyp92]*, pp. 11-22.

[WiL92] U.K. Wiil, and J.J. Leggett, "Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems", in *[Hyp92]*, pp. 251-261.

[ZhP92] Y. Zheng, and M.-C. Pong, "Using Statecharts to Model Hypertext" in *[Hyp92]*, pp. 242-250.