HyperStorM: An Extensible Object-Oriented Hypermedia Engine

Ajit Bapat, Jürgen Wäsch, Karl Aberer, Jörg M. Haake

Integrated Publication and Information Systems Institute (GMD-IPSI)
GMD – German National Research Center for Information Technology
Dolivostr. 15, D–64293 Darmstadt, Germany
Tel.: ++49-6151-869-{960, 959, 935, 918}
e-mail: {bapat, waesch, aberer, haake}@darmstadt.gmd.de

ABSTRACT

It is a well-known problem that developers of hypermedia applications need assistance for modeling and maintaining application-specific hypermedia structures. In the past, various hypermedia engines have been proposed to support these tasks. Until now, hypermedia engines either provided a fixed hypermedia data model and left extensions to the hypermedia application or they left the modeling of the hypermedia data completely to the application developer and only provided storage functionality which had to be plugged into the data model by the application developer. As an alternative, we propose an extensible object-oriented hypermedia engine which supports the specification of application semantics as application classes within the hypermedia engine, thereby supporting complex operations maintaining application-specific as well as application-independent constraints.

In the HyperStorM hypermedia engine, the storage layer and the application layer of a hypermedia system are implemented within the object-oriented database management system VODAK. Only the presentation layer is kept outside the OODBMS. This approach facilitates both the reuse of database functionality as well as the flexibility necessary to support the efficient development of different kinds of hypermedia applications. First evaluations show that our approach presents a much more powerful hypermedia engine than previous approaches, thus giving a new quality to hypermedia application development.

KEYWORDS

Hypermedia engine, open extensible hypermedia systems, database management system support for hypermedia applications

INTRODUCTION

It is a well-known problem that developers of hypermedia applications need support for modeling and maintaining application-specific hypermedia structures. In order to provide such support, the concept of a hypermedia engine has been developed [4, 23]. A hypermedia engine is an abstract machine that provides an interface for accessing and manipulating hypermedia structures. To make the task of application development easier, it should be as simple as possible to express the application-specific hypermedia data model in terms of the hypermedia engine's functionality. Furthermore, to ease the task of maintaining application-specific constraints it is useful to express as much as possible of the application-specific objects, operations, and constraints in the hypermedia engine. In this case, the hypermedia engine would need to provide not only simple load/store functionality but also more complex operations that maintain the application's semantics.

In the past, a number of hypermedia engines have been proposed that provide abstract hypermedia data models and storage functionality: HAM [4], GMD-IPSI's HyperBase [23], Aalborg's HyperBase [32], Hyperform [31], HB3 [15], to name some of them. Most of these hypermedia engines have mainly focused on providing a fixed hypermedia model. Other systems leave the modeling of the hypermedia data completely to the application developer and only provide storage functionality which has to be plugged into the data model by the application developer.

Therefore, it has always been the task of the application developers to map their application-specific data model onto the abstract data model provided by the hypermedia engine. And in addition, they had to implement application-specific objects, operations, and constraints as part of the application program running on top of the hypermedia engine. A problem with this approach is that because of the semantic gap between the application's data model and the hypermedia engine's data model complex operations maintaining application-specific constraints could not be supported by the hypermedia engine.

In this paper, we propose an extensible object-oriented hypermedia engine which supports the specification of application semantics as application classes within the hypermedia engine, thereby supporting complex operations maintaining application-specific as well as application-independent constraints. In addition, the reuse of already exist-

ing/previously implemented hypermedia functionality can be facilitated.

Meanwhile, there is a common agreement that among the different kinds of database management systems (DBMS), object-oriented DBMSs (OODBMS) are the best choice to be used for the storage layer of hypermedia systems [14]. Previous approaches implemented the three layers of hypermedia systems (i.e., according to [4], the storage layer, the application layer, and the presentation layer) as separate modules/processes. New developments of DBMS technology make it possible to implement a hypermedia engine as an extension of an OODBMS. Our approach is to implement most of the hypermedia functionality within the OODBMS VODAK [27]. VODAK's data model is flexible enough to allow the definition of new data modeling primitives. This flexibility is achieved through a meta data layer.

In the HyperStorM[1] hypermedia engine proposed in this paper, the storage layer and the application layer are implemented within VODAK. Only the presentation layer is kept outside the OODBMS. This approach facilitates both, the reuse of database functionality such as persistent storage, concurrency control, crash recovery, and query functionality, and the flexibility necessary to support different kinds of hypermedia applications. The latter is achieved by the inherent means for extending as well as customizing and tailoring existing classes of the hypermedia engine. Thereby, this approach eases the tasks of the application developer and leads to a faster application development.

The paper is organized as follows. In section 2, we present an overview of related research work. Section 3 analyzes the requirements for a hypermedia engine from an application developer's point of view. This leads us to the identification of deficits of current approaches. In section 4, we describe the basic hypermedia model provided by our extensible hypermedia engine, followed by a description of implementation details in section 5. Section 6 gives an example of how to exploit our generic approach to develop a hypermedia-based environment for systems engineering. We finish with a brief summary, conclusions, and a look at future work.

RELATED WORK

Since the emergence of the concept of hypermedia engines several approaches have been suggested. Some (HAM, HyperBase (GMD-IPSI), and HyperBase (Aarlborg University)) have been labeled as "hyperbases of the HAM generation" [31] since these approaches have main concepts in common with the HAM. Several other approaches have emerged, too, introducing different concepts. In this section, we will take a look at some of the most important ones with respect to the task of application development.

HAM. The HAM (Hypermedia Abstract Machine, [4]) has been used to support hypertext-based CAD and CASE applications. However, it was designed to provide sufficient generality for use with other applications. This led to a very low-level storage engine leaving the definition and implementation of all application-specific functionality and the integration of application-level design decisions into the database server to the application developer. The HAM's data model provides five different objects: A graph as the highest level object can contain the other HAM objects (contexts, nodes, links, and attributes). Contexts are used to divide the data within graphs. Nodes may contain arbitrary contents. Links define relationships between nodes. Contexts, nodes, and links may have attributes with arbitrary attribute values which can be used to describe application specific semantics to HAM objects.

HyperBase (GMD-IPSI). GMD-IPSI's HyperBase approach [23] aimed at supporting hypertext-based authoring systems. It provides the hypertext application developer with an application interface to the hypermedia engine. While the storage of persistent objects is handled by the engine, the application developer has to define the application-dependent semantics of those objects in the application program. The data model offers nodes, links, composite objects, and attributes. Nodes, links and composites may carry application-defined attributes. Nodes may have content. Links connect two HyperBase objects, where nodes as well as links and composites are HyperBase objects. Composites are collections of HyperBase objects. Hyperbase and its successor CHS [22] provided multi-user access and transactions.

HyperBase (Aalborg University). Aalborg University's HyperBase [32] is a layered system providing three layers: basic entities (a simple hypertext model with nodes and links), basic services (basic operations to be performed on the basic entities, like e.g. creating nodes and links and connecting nodes with links), and multi-user services (e.g., simple user-controlled locking mechanisms to support asynchronous collaborative work). The basic entities form the data model: Links are separate objects which can only refer (unidirectionally) to nodes. Each link stores its destination node and each node keeps a list of its outgoing links. While nodes are versioned, links are not.

Hyperform. The successor to Aalborg University's HyperBase, Hyperform [31], provides a set of built-in classes which can be used to enrich a self-defined hypermedia data model with DBMS functionality. There are three basic classes (Meta Class, System Object, and Object) and five subclasses of Object. These five classes provide basic functionality for hypermedia applications: concurrency control, notification control, access control, version control, and query and search. Using subclassing, the application developer creates the classes he needs in order to model the application's data model and can then — by means of multiple inheritance — enrich the data model with the functionality provided by the five subclasses of Object, thus adding concurrency control, etc. to the model.

HB3. Within the System Prototype 3 (SP3) at Texas A&M University, HB3 [15] is designed to meet the storage needs of the SP3 which supports process-oriented hypermedia models like e.g. the Dexter model [12]. HB3 has two main components: The association set manager (ASM) and the versioned object manager (VOM). The system's hypermedia data model itself is defined outside of HB3 within the link services manager (LSM), which is located on top of HB3 within SP3's architecture. The VOM is responsible

1. HyperStorM is an acronym for **Hyper**media Document **Stor**age and **M**odeling

for the versioning of the hypermedia objects. Using services of the VOM, the ASM provides the persistent and sharable storage of the hypermedia structure and is aimed to be generic enough to provide storage for basically any data model that describes objects in terms of behavior, structure, and data. Within SP3, the LSM provides one such model. HB3's DBMS functionality resides within the underlying DBMS that is accessible to HB3 via a storage manager (SM).

DeVise. The DeVise hypermedia system prototype [9] follows the Dexter model [12] of a layered architecture. An object-oriented database server provides persistent storage for hypermedia objects while a so-called run-time process (RP) in the run-time layer provides generic and specific storage classes whose instances are stored by the underlying server. The object-oriented database server provides notification and locking mechanisms that can be used by the RP to monitor the creation, deletion, and updating of hypertext elements (nodes, links, composites).

TecPad. In the context of the ESPRIT project TecPad the OODBMS O_2 [6] was used to provide a storage module for hypertext documents [5]. This approach does not provide a hypermedia engine with a well-defined interface to hypermedia applications. It merely offers a mapping of SGML [8] onto classes in O_2 and an extended O_2SQL query language as a means to access or manipulate the hypertext documents stored in the database.

These hypermedia engines either provide a rather fixed hypermedia data model that can only be extended by adding attributes whose semantics are not maintained by the engine and therefore have to be handled by the hypermedia application ("HAM generation") or they only provide storage functionality which can be plugged into a hypermedia data model that has to be specified by the application developer (Hyperform, HB3, and DeVise).

In the first case, this leads to a large semantic gap between the application's data model and the hypermedia engine's data model that has to be overcome by the application developer and in the second case the application developer is bothered unnecessarily with decisions about the persistent storage of the hypermedia application he is to develop. Therefore, we now focus on the requirements of an extensible object-oriented hypermedia engine that eases the tasks of hypermedia application developers.

REQUIREMENTS ANALYSIS

Hypermedia applications are diverse in nature. To identify the requirements for a hypermedia engine, we took a closer look at some state-of-the-art hypermedia applications such as Aquanet [17], MacWeb [20], StorySpace [3], and SEPIA [24]. The following general requirements are the result of this analysis. Each requirement is denoted by a number (e.g., R1.1) for further referencing throughout this paper.

First, a hypermedia engine must provide an abstract hypermedia data model that is a good foundation for modeling application-specific hypermedia structures (R1).

- (R1.1) The abstract hypermedia data model should provide a set of basic types of hypermedia objects. Usually, they include atomic and composite nodes, links, contents, anchors, etc. In addition, some applications require extensions, e.g., links that may connect not only nodes but also point to other links (see, e.g., the modeling of Toulmin schemata in SEPIA). For other applications it is required that anchors can also be placed inside the content of nodes and links (e.g., videos in [7]).
- (R1.2) The abstract hypermedia data model should support the definition and maintenance of structural constraints among hypermedia objects. For example, constraints may relate to, e.g., composites that may not contain themselves recursively or links that must always have defined endpoints.
- (R1.3) The abstract hypermedia data model has to offer complex operations for the manipulation of hypermedia documents that maintain these constraints.
- (R1.4) The generic hypermedia data model should be able to integrate external storage systems such as video servers and document management systems.
- (R1.5) To support open hypermedia systems, the generic hypermedia data model should be able to integrate external tools and external data generated by such tools.

Second, a hypermedia engine must provide means for expressing application semantics (R2).

- (R2.1) The hypermedia engine must support the specification of application-specific kinds of objects. This can be used to implement typed hypermedia objects as, e.g., in Aquanet, MacWeb and SEPIA. The semantics of these classes include their internal structure (static aspect), e.g. typed attributes, their dynamic behavior such as operations, and constraints.
- (R2.2) The enforcement of constraints defined by the application's types of hypermedia objects has to be guaranteed by the hypermedia engine. Such constraints may relate to, e.g., cardinality, connectivity of links, and structural properties (e.g., objects required in composites or acyclic hypermedia networks).
- (R2.3) Different roles for hypermedia objects must be supported, since many hypermedia applications support multiple use of hypermedia objects in different contexts with different behavior. Examples are Aquanet's relations and SEPIA's activity spaces.
- (R2.4) Role transformations must be supported by the hypermedia engine in order to use hypermedia objects in different contexts without manually adapting their roles.

Third, the re-use of (parts of) *existing* hypermedia applications should be supported so that the development of *new* hypermedia applications can benefit from existing ones (R3).

- (R3.1) The hypermedia engine should allow the re-use and tailoring of application-specific types of objects among different applications.
- (R3.2) The hypermedia engine should ease the integration of application-specific types of objects.

Fourth, a hypermedia engine should provide support for common functionality that may be used by many different hypermedia applications (R4). Thus, a hypermedia engine has to support:

- (R4.1) persistent storage,

- (R4.2) multi-user access incl. concurrency control,
- (R4.3) crash recovery,
- (R4.4) integration of multimedia data types,
- (R4.5) declarative access to hypermedia structures (query and search functionality),
- (R4.6) access control, and
- (R4.7) client/server distribution.

For the use in multi-user, collaborative applications working on shared hypermedia documents, the hypermedia engine should provide (R5):

- (R5.1) versioning of hypermedia documents and
- (R5.2) mechanisms to communicate group-awareness.

When looking at the current approaches (cf. "Related Work") in search for a hypermedia engine that would meet our requirements, we had to realize that the current approaches all have some deficits with respect to our list of requirements.

Systems of the HAM generation do not provide a hypermedia data model that is flexible enough for our needs. The definition of structural constraints that go beyond the rather simple data model that is provided can only be achieved by adding attributes whose semantics can not be handled by the hypermedia engine and, thus, have to be maintained by the application (R2.2). For Hyperform, the whole data model has to be defined by the application developer. All of the current approaches limit the operations offered to basic functions like reading, writing, creating, and deleting basic objects, i.e. links and nodes and do not offer any complex operations (R1.3) The DeVise system leaves the modelling of the hypermedia data to the programmer of the RP, and the TecPad approach only provides a rigid mapping of SGML DTDs onto O_2 classes.

The specification of application semantics within the hypermedia engine (R2) is not provided for by HAM generation of systems. These system explicitly expect application semantics to be handled by the application. Within Hyperform, the application developer can transfer application semantics into the hyperbase since he defines the hypermedia data model on his own, but he is not provided with any ready-to-use mechanisms for this task. Roles as well as role transformation (R2.3 and R2.4) can not be handled within the hyperbases of the HAM generation, too. For SP3, the application developer has to go outside HB3 because the hypermedia data model resides above HB3 (within the LSM). In DeVise, application semantics have to be implemented within the RP using the notification and locking mechanisms of the underlying OODB server.

The re-use of existing hypermedia applications (R3) is only supported by Hyperform and, to a certain extent, by HB3. In the latter system applications that use the same hypermedia data model can re-use the LSM.

All current approaches offer persistent storage support (R4.1) — in some cases (e.g., HAM) by simply implementing file server functionality — but all other requirements of the fourth group (R4) are if at all, only covered incompletely: The HAM offers access control lists that can but need not be attached to objects in order to support multi-user access (R4.2). None of the systems can provide full DBMS functionality (including crash recovery (R4.3) and query functionality (R4.5)) since none really incorporates a DBMS. HB3 can only make use of the services of the underlying DBMS but can not integrate them with the hypermedia data model because the latter is residing in a different layer of the system's architecture. Due to its object-oriented concepts, Hyperform does offer means to integrate or even extend predefined DBMS functionality but it does not force hypermedia applications to do so, thus leaving potential for inconsistencies when hypermedia applications that do not use the DBMS functionality operate on data that is also used by applications that rely on mechanisms like, e.g., concurrency control. DeVise supports multi-user access and concurrency control (R4.2). The TecPad approach can only be regarded as a storage back-end for hypertext systems but not as a hypermedia engine with any additional functionality. Access control (R4.6) is offered by all systems only in the context of concurrency control (locking) but not in the sense of of access rights for individual users. All systems support client/server distribution (R4.7).

The HAM, Aalborg's HyperBase, Hyperform, and HB3 provide rudimentary versioning mechanisms: Versions of objects can be created and deleted but there is no further functionality for the management of versions, e.g. merging (R5.1). Although some systems (Aalborg's HyperBase, Hyperform, DeVise) offer event notifications there are no mechanisms available that can be used straightforward to communicate group-awareness (R5.2).

As a consequence, there is currently no approach that would cover all of the requirements listed above. Since these requirements are derived from state-of-the-art hypermedia systems, hypermedia application developers would benefit most from a generic, extensible hypermedia engine that matches the requirements listed above. In the next section, the design of the abstract hypermedia data model of our hypermedia engine prototype is introduced and its relationships to the above requirements is discussed.

DESIGN OF THE HYPERMEDIA ENGINE

In this section we describe the basic hypermedia data model provided by the HyperStorM hypermedia engine.

The VODAK data modeling language

Because there exist conceptual and terminological differences between different OODBMSs, the relevant concepts of the used OODBMS VODAK [27, 13] and its data modeling language (VML) are introduced briefly in the following.

As in other OODBMS, objects are used to represent material or immaterial entities, or abstract concepts. Objects are identified through unique object identifiers (OID). The structure (properties) and procedural behavior (methods) of objects is described through abstract data types which are called *object types*. VODAK distinguishes between object types and *classes* (dual model). Object types determine the structure and behavior of objects and, hence, are intensional, whereas class definitions describe class objects which act as containers for their instances (extension of a class).

A class definition consists basically of two parts: the object type of the class object itself (`OWNTYPE`) and the object type

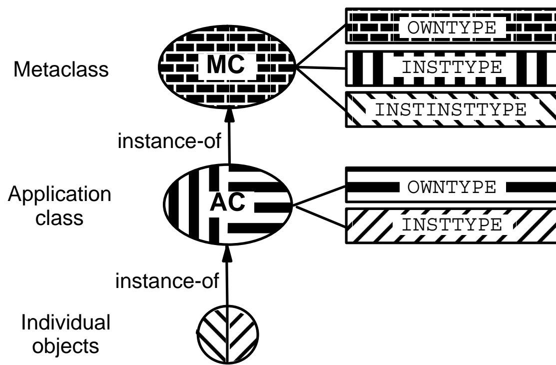

Figure 1: Classes and type composition in VODAK

of the class' instances (`INSTTYPE`). The initialization part (`INIT`) allows initialization of the class' properties by calling methods of the class' own type.

In VODAK, classes are first-class objects, i.e., they can be treated like ordinary objects. Thus, it is possible to create classes and modify classes' properties *at run-time*. Because classes are treated as first-class objects, class objects are themselves instances of other classes, called *metaclasses*.

Metaclasses are used in VODAK to describe the common structure and behavior of application classes *and* their instances which may not be known at the time the metaclass is defined. The `INSTTYPE` associated with a metaclass determines the common semantics of the metaclass' instances, i.e., its application classes. The metaclass' `INSTINSTTYPE` defines the common semantics of the instances of its application classes, i.e., individual objects. Thus, the semantics of an individual object is determined through the `INSTTYPE` associated with the object's class and the `INSTINSTTYPE` associated with the object's metaclass (see Figure 1).

Design approach

In the following we give a brief overview of the design process of the hypermedia engine. Figure 2 indicates the design steps described below.

Meta modeling. One reason to use metaclasses is to model semantic relationships between application classes. Many OODBMS offer hard-coded mechanisms to describe such relationships. But semantic relationships can have several dimensions. Thus, only a flexible mechanism like freely definable behavior and integrity constraints for metaclasses allows to model a great variety of dedicated semantic relationships, as needed for our hypermedia engine semantics. Metaclasses ensure the consistent usage of the object types defining the semantic relationships and enforce specific integrity constraints by declaring an application class as an instance of a specific metaclass.

For defining the abstract hypermedia data model we had to provide basic semantic relationships. We have identified the following four orthogonal semantic relationships (which are discussed in the next section) to be sufficient:

- Flat hypermedia structures (e.g., different kinds of links in a hypermedia network)
- Element- and set-association (e.g., for modeling composite nodes)

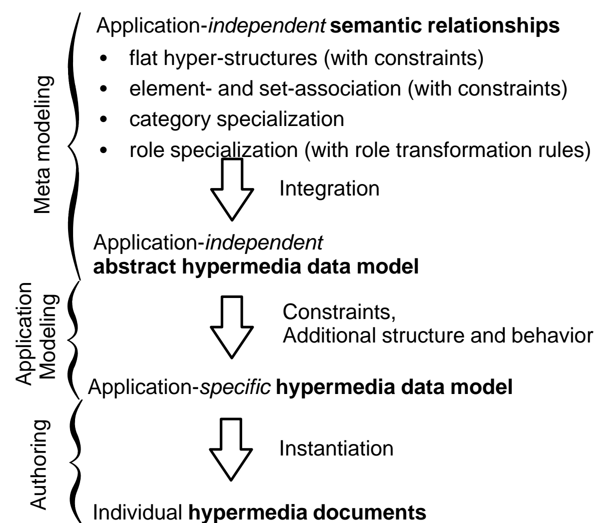

Figure 2: Overview of the approach followed in the design of the hypermedia engine.

- Category specialization (e.g., for modeling links that can behave like nodes)
- Role specialization (e.g., for modeling roles of objects and role transformations)

These semantic relationships enforce the structural constraints required by (R1.2), and include constraint-based and rule-based facilities that can later be used to tailor the abstract hypermedia model to the needs of a specific application (R2). The four basic semantic relationships are used to define a set of metaclasses describing the application-independent abstract hypermedia data model and extending the OODBMS VODAK to the HyperStorM hypermedia engine. These metaclasses constitute extensions of VODAK's data model that provide the necessary modeling primitives for the development of dedicated hypermedia applications and ensure the consistent usage of the semantic relationships.

Application modeling. Up to now, the data modeling takes place at the meta level and does not consider application-specific hypermedia semantics. The abstract hypermedia data model can now be tailored by an application designer to the needs of a specific application, i.e., defining the application-specific hypermedia semantics. This is done by classifying objects into *application-specific* types of hypermedia objects (R2.1) and adding constraints with respect to the semantic relationships (R2.2, R2.4).

In VODAK, the application-specific types of hypermedia objects are defined by application classes (R2.1) which are instances of appropriate metaclasses describing the basic semantics for hypermedia objects. To specialize these semantics it is possible to add constraints and transformation rules to the application classes (R2.2, R2.4) and to use additional VODAK object types in the definition of application classes (R2.1). This extension of structural and behavioral semantics of hypermedia types is not possible if types of hypermedia objects are simply represented by means of attribute values (strings) as in [4, 23, 32]. Modeling the specific semantics of a hypermedia application is further discussed in the section "Tailoring the abstract hypermedia model".

This modular assembly concept used in the design of the hypermedia engine enables a developer of an application-specific hypermedia model to re-use existing VODAK object types and class definitions (R3.1). Moreover, he or she can refine and change the model's semantics at run-time which results in reduced development time of the application-specific hypermedia model. At the same time, the hypermedia engine ensures the generic and application-specific integrity constraints without programming efforts. Moreover, the integration of existing application classes is eased because they are based on a common abstract hypermedia data model (R3.2).

Semantic Relationships

In the following we explain the basic semantic relationships shortly. For a detailed formal description we refer the reader to [28].

Flat hypermedia structures. In the simplest case, a flat hypermedia structure can be viewed as a *graph* consisting of nodes and binary (directed and bidirectional) links. Because we also allow links to refer to other links we extend the definition of graphs by means of a recursive definition of structures that allow links connected to links (R1.2), links connected to links that themselves connect links and so forth (a precise mathematical definition is given in [28]). Most hypermedia systems prohibit loops and dangling links to achieve well-formed hypermedia documents. Therefore, we consider the following application-independent constraints for hypermedia structures:

- Loops, i.e., links where the source and destination are identical, are not allowed in the (generalized) graph.
- Dangling links, i.e., links with undefined source and/or destination, are not allowed (R1.2).

We allow the usage of typed links and nodes (R2.1) in the (generalized) graph with the additional constraint:

- Multiple links between the same two objects are allowed only if the links have different types.

To model constraints on the connectivity of links and nodes, as required by (R2.2), we introduce a constraint function $cons_{\text{link}}$ that determines if a link of a specific type is allowed to connect two typed objects.

- A link l can connect two objects o_1, o_2 only if $(type(o_1), type(o_2)) \in cons_{\text{link}}(type(l))$.

Element- and Set-Association. In accordance with the Dexter hypermedia reference model [12], we decided to separate the nodes and links from their contents (Dexter within-component layer). The internal structure of the content objects are described separately from the hypermedia structure (see Figure 3). Thus, at the hypermedia abstraction level content objects can be viewed as atomic.

To group these atomic content objects, e.g., into a node, we use the concept of *element-association* [18]. Element-association introduces a set object to describe properties of a group of element objects. To model composite objects we use the concept of *set-association* [18]. Set-association introduces a superset object in order to describe properties of a group of subset objects. It may be applied recursively, building an n-level hierarchy. Therefore, we can view com-

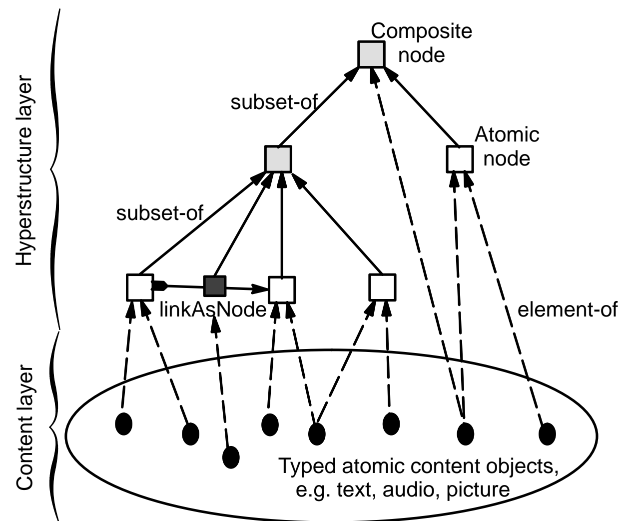

Figure 3: Example of the usage of element- and set-association.

posite nodes as supersets which are composed out of other (atomic or composite) nodes, links and atomic content objects. The only application-independent constraints for our concept of set-association is as follows:

- The set objects related by set-association must correspond to a tree structure (R1.2).

Because it is useful to share atomic content (e.g., text, pictures, audio) objects we do *not* enforce this constraint in the case of element-association. Links, atomic and composite nodes (which are modeled as set objects) can be shared among composites using the concept of role specialization described below.

We combined a typing and constraint mechanism (R2.2) with the association concept for tailoring the hypermedia model to application-specific needs, similarly as with nodes and links.

- An association *a* can contain only elements or sets as specified $cons_{\text{ass}}(type(a)) = \{(t_1,min_1,max_1), ..., (t_n,min_n,max_n)\}$, i.e., objects of type t_i with minimum min_i and maximum max_i occurrence. Objects are created *automatically* if necessary (minimum occurrence).

For instance, this constraint mechanism can be used to express that in SEPIA's argumentation space only nodes and links regarding the Toulmin schema are allowed to exist or to express that a specific node has exactly one text object assigned (see also example 2).

Category specialization. Specialization captures the intensional concept of subtyping and the extensional concept of extension inclusion between classes. We assume the following consistency constraints on the classes as well as on the instances of these classes, that participate in the category specialization relationship:

- The *specialization-of* relation must be a partial order. Additionally we allow for a subclass only one direct superclass. Thus, we obtain a tree of classes (single inheritance).

- The extension of the specialization class is a subset of the extension of its generalization class.
- The extensions of two specialization classes of one generalization class has to be disjoint.

Inheritance of properties and methods is an important characteristic of the specialization relationship. If an object or class receives a method that is not part of its interface, this method together with the arguments is delegated to the more general object (inheritance at the level of individual objects) or to the superclass, respectively (inheritance at the class level).

Category specialization is used in various ways within our hypermedia engine. For example, in the Toulmin schema certain links can behave almost like nodes, as they may be referred to by other links or in the MUSE application [16] links may have textual content objects assigned. Thus, a specific link class might be modeled as a specialization of a node class.

Role specialization. As mentioned in the requirements section, nodes and links can occur as different roles in different contexts, e.g., composites (R2.3). The hypermedia engine should be able to keep track of such a relation between objects. Some properties of the two object roles are shared, but there exist also some property values that differ.

To model this, we use the concept of *role specialization*. Role specialization is similar to category specialization, except that the last property of category specialization needs not to hold. The general object contains the shared properties, whereas the different roles contain the non-shared properties. In addition to that, the role objects can have different structure and behavior, defined by their own object types. The role objects are related by *role-of* relationships with the general objects.

To specify the possible role transformation of hypermedia objects (R2.4) we introduce a role transformation function which specifies the set of classes which can appear as a role of a general object. We assume that the object's state does not change during the transformation. Details on role modeling and role transformations can be found in [13, 28].

Defining the abstract hypermedia model

Semantic relationships are combined to describe the metaclasses defining our abstract hypermedia data model. When combining semantic relationships, additionally the *inter*-semantic-relationship constraints have to be considered. It is not enough to inherit the definitions of semantic relationships via subtyping. The semantic relationships must be integrated in a meaningful way to provide complex, consistency-preserving operations (R1.3) while maintaining application-independent (R1.2) and application-specific consistency constraints (R2.2).

Example 1: Remember that with respect to source and destination of links different hypermedia applications may define different constraints: some systems like SEPIA [24] restrict them to objects contained in the same composite whereas other systems like StorySpace [3] also support links between objects contained in different composites (tunnel links). To define an operation to create a link between two objects (see also the system architecture section), we first have to check if the link of the given class is allowed in the composite (association constraint) and if the source and destination objects are contained in the same composite (if the specific link class does not specify tunnel links). Then, (in both cases) we have to check if the link of the given class is allowed to connect the given objects (link constraints). If all of these constraints are fulfilled we can create the link object (including its connection to the source and destination objects) and finally add the link to the composite object.

The combined semantic relationships are used to define the metaclasses constituting the HyperStorM hypermedia engine. In the current implementation, the following metaclasses provide basic hypermedia objects of the kind required by (R1.1), including complex operations that maintain consistency of the hyperdocuments (R1.2, R1.3).

- **Nodes:** AtomicNode, CompositeNode, VirtualCompositeNode, Node
- **Links:** DirectedBinaryLink, BidirectionalBinaryLink
- **Composite contents:** CompositeContent, Organizer
- **Atomic contents**: AtomicContent, BytestringAtomicContent, ExternalReferenceAtomicContent

The Node and CompositeContent metaclasses together allow the modeling of objects that can behave like atomic *or* composite hypertext objects. The VirtualCompositeNode metaclass models composites that are defined using VQL queries [1] and can be used to provide different views on hypermedia networks (R4.5). Organizer classes are used to organize complete hyperdocuments in a directory-like way. The AtomicContent metaclass is used to implement application-specific classes that model different kinds and formats of media within VODAK (R4.4). The BytestringAtomicContent and ExternalReferenceAtomicContent metaclasses support storage of multimedia data (e.g., text, pictures, audio, video) as BLOBS in VODAK or in external storage systems like EOS [2], respectively (R1.4).

The last two of the above metaclasses also include generic mechanisms for the integration of external applications (R1.5). This way external tools can be used to view and edit objects that are managed by the hypermedia engine (internal data as well as external data). Any integration going beyond this is not covered by the current implementation. Also local anchoring as mentioned in (R1.1) is not yet supported.

Tailoring the abstract hypermedia model

The metaclasses shown in a previous section are used by a application designer to tailor the abstract hypermedia model to the needs of a specific application domain, thus, defining the application-specific hypermedia data model. Application classes simply are declared as instances of an appropriate metaclass to provide them with the intended behavior.

The application designer can assert constraints regarding the semantic relationships to the application classes (R2.2, R2.4). Initialization of constraints is done at class creation time by executing the initialization clause (`INIT`). The constraints can be changed at run-time because constraint insertion and deletion is done by ordinary method calls. Moreover, it is possible to create new classes as instances of an ex-

```
CLASS isBasedOn METACLASS DirectedBinaryLink
   // declaration as an instance of a metaclass
   INIT SELF->specializationOf(linkAsNode);
      SELF->isTunnelLink(FALSE);
      SELF->addLinkConstraints(
         [Component, Requirement],
         [Specification, Requirement]);
      SELF->addElementConstraints(
         [TextContent,0,1],
         [AudioContent, 0, 1], );
      SELF->addTransformationRule(
         ModelingSpace,ValidationSpace,
         isDerivedFrom);
      SELF->addTransformationRule(
         ModelingSpace, Variant, isBasedOn);
      . . .
END
```

Figure 4: Example of tailoring an application-specific class by adding constraints.

```
CLASS AudioContent METACLASS AtomicContent
   OWNTYPE VODAK_Audio_ClassType
      // adding  additional VODAK object types
      // that model audio class' and
   INSTTYPE VODAK_Audio_Type
      // audio instance's structure and behavior
END
```

Figure 5: Example of adding application-specific structure and behavior to a class.

isting metaclass at run-time because classes are treated as first class objects in VODAK.

Example 2: An example based on the MUSE-specific hypermedia data model [16] (see also section "Applicability") is shown in Figure 4. The class isBasedOn is declared as an instance of the metaclass BinaryDirectedLink which describes the general behavior of this kind of links (including the role mechanism described above). By means of category specialization it is declared as a specialization of the class linkAsNode which is an instance of the metaclass AtomicNode and therefore inherits the node-like behavior to link instances of class isBasedOn. Moreover, application-specific constraints on the connectivity ($cons_{\text{link}}$(isBasedOn) = {(Component, Requirement), (Specification, Requirement)}) and the possible content of the link ($cons_{\text{ASS}}$(isBasedOn) = { (TextContent, 0, 1), (AudioContent, 0, 1) }) are asserted to the class. Role transformation rules are added which express, e.g., that if a link of class isBasedOn is copied from an ModelingSpace composite to a ValidationSpace composite, it is automatically converted to a link of class isDerivedFrom.

Moreover, it is possible to provide additional structure *and* functionality for an application class, that is instantiated from one of the abstract hypermedia model's metaclasses, by adding (pre-existing) application-specific object types in the class definition (R2.1, R2.3, R3.1, R3.2).

Example 3: The class AudioContent (see Figure 5) is defined as an instance of the metaclass AtomicContent which describes the general semantics of content objects in our hypermedia engine. Additionally, an `OWNTYPE` and an `INSTTYPE` are defined for this class and its instances, which provide the properties and methods for storing and manipulating audio objects within VODAK. Therefore, the structure and behavior is composed of the abstract semantics inherited by the metaclass *and* the application-specific object types. There are no constraints assigned to the class AudioContent because constraints regarding the element-of relationships are assigned to the set objects.

SYSTEM ARCHITECTURE OF THE HYPERSTORM HYPERMEDIA ENGINE

In this section we describe the system architecture of the implemented HyperStorM hypermedia engine. The overall system architecture is shown in Figure 6.

The core of the HyperStorM hypermedia engine consists of the object-oriented database management system VODAK. VODAK contains the metaclasses that implement the basic hypermedia engine functionality, e.g., the checking of generic and application-specific integrity constraints and consistency-preserving operations for the manipulation of hypermedia structures. Moreover, VODAK stores and manages in its data dictionary the application-specific hypermedia models described by application classes, their application-specific constraints, and additional object types.

External storage systems, e.g., EOS for storing large videos, can be connected to VODAK by instantiating the ExternalReferenceAtomicContent metaclass. Accesses and manipulations to these external storage systems are managed by the HyperStorM hypermedia engine, and, hence, are transparent for applications running on top of it. The invocation of external application programs is handled by the hypermedia engine, too.

Applications of the HyperStorM hypermedia engine are implemented using the C++ based VODAK client interface (R4.7). The VODAK clients may run on arbitrary nodes in the network and communicate via the VODAK server interface with the hypermedia engine. Basically, the client interface can be considered as a remote API to the VODAK OODBMS; it offers VML data types and the VODAK query language (VQL) that can be used to build applications programs like graphical user interfaces. Moreover, it offers support for visualization and manipulation of multimedia data stored within VODAK (R4.4). Client applications communicate with VODAK via a generic interface which consists of the following functions:

- getting the OID of a class by sending the class' name;
- begin, commit and abort of a VODAK top-level transaction;
- submitting arbitrary VML method calls to VODAK and transferring back the results;
- submitting declarative VQL queries to VODAK and transferring back the results.

The complex, consistency-preserving operations to manipulate hypermedia documents offered by the VODAK hypermedia engine are invoked by the client applications using the method call interface. Each of those operations is executed as a single VODAK top-level transaction by default. Using the transactional commands offered by the interface, an application programmer can build new complex transactions,

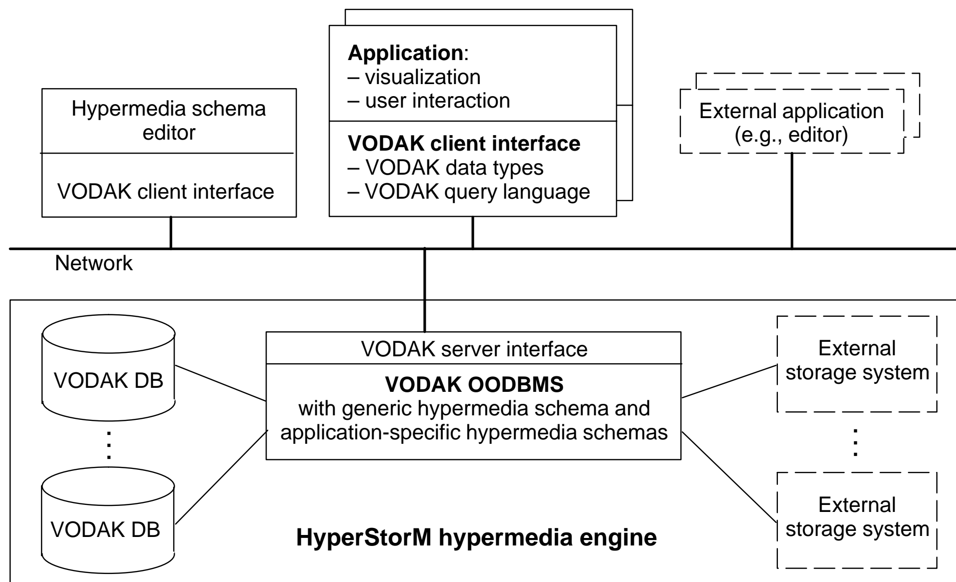

Figure 6: System architecture of the HyperStorM hypermedia engine.

e.g., macros, consisting of several consistency-preserving operations. Utilizing VODAK's open nested transaction model [19] and the commutativity predicates defined for the hypermedia engine's operations, each operation of an application-defined transaction can be executed as a subtransaction, increasing the degree of concurrency without loss of ACID properties. Moreover, an application programmer can use declarative VQL queries (including the hypermedia engine's operations), thus, enabling set-oriented access to hypermedia documents (R4.5).

From the above discussion it should be clear that the hypermedia applications (located in the clients) only keep information that is necessary to visualize the hypermedia network and to interact with the user (e.g., coordinates of nodes and links, menu information). This information is delivered by the complex operations provided by the hypermedia engine. All other information (for example, contents, constraints, details about associated external applications) is needed only within the hypermedia engine. Below, we present an excerpt of operations provided by the engine to give an impression of how extensive the engine's functionality is.

- `createNode(nodeClass:OID,composite: OID,name:STRING,position:POINT):OID`
  To create a node of a specific application class (nodeClass) the client calls this interface function. The engine checks whether the node class is allowed within the composite's class. Thus, the client does not have to do any constraint checking (as mentioned earlier, all constraint checking is done by the engine): the engine either returns the object identifier of the newly created node or informs the client that due to the constraints the node could not be created.
- `deleteNode(node:OID)`
  This function not only deletes the node itself but also any dangling links, even recursively if there are links referring to deleted links!
- `createLink(linkClass:OID,composite: OID,source:OID,destination:OID):OID`
  This operation either returns the object identifier of the newly created link or a notification that creation was not successful due to constraint violation (see example 1 in the previous section).
- `copyObjectsToClipboard(objects: {OID})`
  For each user, the hypermedia engine maintains one clipboard which is stored persistently in the database. The copy operation copies the specified objects into this clipboard.
- `pasteObjectsIntoComposite(composite: OID):PASTE_INFORMATION`
  This operation pastes the contents of the user's clipboard into the specified composite. Before doing so, type transformations are performed (recursively). All transformed objects that do not violate the type constraints are then actually created. The operation returns information needed to update the client application.
- `createAtomicContent(object:OID,atomicContentClass:OID,name:STRING):OID`
  This function returns the object identifier of the newly created atomic content. In the case of an external content object, the reference to a new object in the external storage system is established within the hypermedia engine.
- `openAtomicContent(object:OID,atomicContent:OID)`
  This operation opens (on the client's machine) the external application associated with the atomic content.

APPLICABILITY

In this section we present some experiences and results obtained by an evaluation of the implemented HyperStorM hypermedia engine.

The results of the evaluation are based on a hypermedia engine that is tailored to the application-specific needs of the

MUSE system. The MUSE project[2] [16] is aimed at developing various tools to support the modeling and validation of complex technical systems like for e.g. cars, aircraft, or power stations and to integrate these tools in one common environment — the MUSE system. For the user interface of the MUSE system a hypermedia application has been developed [11]. For the validator who might want to check whether the designed technical system fulfills legal regulations, the system visualizes the diverse components of the modeled technical system and their various relationships with each other. The modeler, on the other hand, can interactively create and redesign the model and its components. Both, i.e., the validator as well as the modeler, can make use of the different MUSE tools like simulation tools, tools for supporting software/hardware codesign, etc.

For the MUSE system, the hypermedia data model that was defined consists of 11 metaclasses and 64 application-specific classes. Modeling the MUSE-specific semantics by these 64 application-specific classes derived from our abstract hypermedia data model took only *one* day for one person who is familiar with the MuSE hypermedia model and the use of the HyperStorM hypermedia engine. And by previous arguments this model ensures the maintenance of the generic and application-specific integrity constraints of the hypermedia model within the DBMS-based hypermedia engine. The various tools of the MUSE project as well as their data were easily integrated (R1.5), e.g. the software/hardware codesign tool uses the same database to store its data which can then be visualized by the hypermedia system.

Of course, we have to raise the question whether this efficiency and flexibility in design and safety in execution can be compliant with reasonable run-time performance. For this reason we performed some preliminary experiments with our DBMS-based hypermedia engine. Performance measurement was done on a SGI Indy workstation running the HyperStorM hypermedia engine. A single client ran on a SUN Sparc-10. The engine operated on hyperdocuments containing more than 10,000 hypermedia objects.

The results of this evaluation are shown in Figure 7. All of the operations shown are covered by transaction management. Operations on attributes are of magnitude 10 ms (dependent on the data type and size of the attribute), operations on single objects (e.g., creation of a link including all constraint checking) of magnitude 100 ms, and operations on whole hypermedia structures and documents take about 1000 ms (depending on the size of the structure). The only other performance measurements in the area of hypermedia engines we are aware of are those of Aalborg's HyperBase and the Hyperform system published in [30]. Compared to these numbers the operations offered by the HyperStorM hypermedia engine are much faster (even though our operations seems to be more powerful).

These results show that an adequate performance for interactive hypermedia systems that are built based on the HyperStorM hypermedia engine is achieved although all the manipulations on the hypermedia structure are done within the hypermedia engine. The response times are such that interactive editing operations suffer no performance problems. Additionally, one has to consider that the numbers were obtained from the first fully functional prototype of the hypermedia engine without particular optimizations.

Operation	Execution time in ms
changeName (of a node)	10
createNode	162
createLink (link with node behavior)	270
createLink (constraint violation – no link created)	111
createAtomicContent	198
copyObjectsToClipboard (5 HM objects (3 nodes, 2 links))	370
pasteObjectsIntoComposite (role transform. of 5 HM obj.)	1070
openComposite (15 HM objects contained in composite)	316

Figure 7: Performance of selected operations offered by the HyperStorM hypermedia engine.

One may ask how our hypermedia engine will scale up in case of a large number of clients. Scalability is a problem in every centralized or client/server (relational or object-oriented) DBMS with light-weighted clients. Since we implemented the HyperStorM hypermedia engine as an extension of an OODBMS, we can profit from research and development in the OODBMS area. However, at the moment it is possible to run several VODAK-based HyperStorM engines in parallel on different machines in the network. Each of them can serve a dedicated set of clients. The only restriction is that they have to share a single storage manager to be able to operate on the same physical database.

CONCLUSIONS AND FUTURE WORK

The objective of this paper was to provide support for the development of hypermedia applications. As a solution for this problem, we proposed an extensible hypermedia engine based on the OODBMS VODAK which supports the specification of application semantics as application classes in the hypermedia engine.

Based on a requirements analysis of hypermedia engines we developed an abstract hypermedia data model that provides the basis for modeling application-specific hypermedia structures. Application developers can use the abstract hypermedia data model to construct application classes that capture the application's semantics in the hypermedia engine. By implementing the hypermedia engine as an extension of an OODBMS, the application developers can use the common functionality provided by DBMSs such as concurrency control, crash recovery, multimedia data types, declarative access to hypermedia objects (query and search), and access control (R4). The separation of the hypermedia engine (i.e., the storage and application layer) and the hypermedia application's user-interface (i.e., the presentation layer) enables the development of different user interfaces for a hypermedia application. Since the hypermedia application's functionality is implemented in the OODBMS (in the ap-

2. MUSE is short for **MU**ltimedia technology for **S**ystems **E**ngineering. The project is sponsored by the Deutsche Forschungsgemeinschaft (DFG), grant number HE 1170/5–1.

plication layer of the hypermedia engine) all complex operations are performed in the DBMS.

Apart from the support for collaboration (R5), local anchoring (part of R1.1), and a tight integration of other hypermedia applications (part of R1.5), our approach meets the requirements introduced in the section "Requirements Analysis" and provides the following benefits:

Providing higher-level modeling constructs in the abstract hypermedia data model eases the task of application development. Application developers are relieved from defining complex hypermedia semantics from scratch as they can adapt the modeling constructs of the abstract hypermedia data model to their needs. For application-defined more complex operations, the mapping of complex operations to the storage model of the hypermedia engine is avoided.

Since the application developers can use application classes from existing applications, application development is further eased by supporting re-use of classes.

Building the hypermedia engine and the applications in a DBMS facilitates the maintenance of consistent hyperdocument structures within and between hypermedia applications. This extends to both, consistency of the abstract hypermedia data model and the application's hypermedia data model. This is ensured for multi-user access — also in the case where different applications access the same hypermedia database, as there is a common data model for different applications — by concurrency control and in case of failure of the system by transaction management. The mechanism to accomplish this is to allow only those possible transactions on the database that implement meaningful and consistency-preserving operations with regard to the application semantics.

While still maintaining a layered approach to hypermedia application development (i.e., abstract hypermedia data model vs. application classes vs. application instances), the integration of these layers in an object-oriented DBMS reduces the number of between-components interfaces and thus reducing the translation overhead involved in mapping concepts of a higher layer to more primitive concepts of a lower layer provided by a different system component. As has been shown in the MUSE system this approach can yield reasonable performance for interactive applications.

Since the application and the hypermedia engine are both modeled in the object-oriented DBMS, all complex operations of the application are performed entirely in the DBMS. Thus, there is less need for implementing complex caching strategies in the application's clients since not so many objects need to be exchanged between application client and hypermedia engine. The latter is especially cost-intensive when clients and server are distributed in a networked environment.

Another benefit of integrating hypermedia applications in a hypermedia engine implemented in a DBMS is the simpler interaction of applications with other DBMS applications and the exploitation of other DBMS services, like advanced multimedia features or declarative querying facilities (R4). The latter can now fully exploit the application's semantics and thus become a more flexible and powerful tool for accessing the hypermedia database [1].

As a result, we can state that the development of hypermedia applications by using an extensible hypermedia engine implemented in an OODBMS is considerably eased. Since other current approaches (as described in section "Related Work") do not cover our list of requirements in such a complete and extensive manner (cf. section "Requirements Analysis"), our approach presents a much more powerful hypermedia engine, thus giving a new quality to hypermedia application development.

At the moment, the HyperStorM hypermedia engine has been used to implement several hypermedia systems within IPSI (e.g., a version of SEPIA and an editor for multimedia presentations) and outside (e.g., MUSE). In the future, we will extend the abstract hypermedia data model depending on the demands of new applications, such as the hypermedia-based meeting support system DOLPHIN [25].

So far, our hypermedia engine can support multiple users working on the same hypermedia database. In the future, we want to extend our approach to include support for cooperative applications (R5) as well [26]. This requires, e.g., the development of new modeling constructs in the abstract hypermedia data model, the extension of current DBMS concepts and technology to support cooperative transaction models [21, 29], and the integration of versioning [10].

ACKNOWLEDGEMENTS

The authors would like to express their thanks to the unknown reviewers of the submitted version of this paper. We especially appreciate Kaj Grønbæk's valuable comments.

REFERENCES

1. Aberer, K. and Fischer, G. Semantic Query Optimization for Methods in Object-Oriented Database Systems. Proceedings of the 11th IEEE Conference on Data Engineering (ICDE '95), Taipei, Taiwan, 1995, pp. 70–79
2. Biliris, A. and Panagos, E. EOS User's Guide, Release 2.2, Technical report AT&T Bell Laboratories, 1994
3. Bolter, J.D. and Joyce, M. Hypertext and Creative Writing. Proceedings of the first ACM Workshop on Hypertext (Hypertext '87), Chapel Hill, N.C., pp. 41–50
4. Campbell, B. and Goodman, J.M. HAM: A General Purpose Hypertext Abstract Machine. Communications of the ACM, Vol. 31, No. 7, July 1988, pp. 856–861
5. Christophides, V. and Rizk, A. Querying Structured Documents With Hypertext Links using OODBMS. Proceedings of the 5th ACM European Conference on Hypermedia Technology (ECHT'94), Edinburgh, UK, September 18–23, 1994, pp. 186–197
6. Deux, O. The Story of O_2. IEEE Transactions on Knowledge and Data Engineering, 2(1):91–108, 1989
7. Geißler, J. Surfing the Movie Space: Advanced Navigation in Movie-Only Hypermedia. Proceedings of the 3rd ACM International Multimedia Conference (Multimedia '95), San Francisco, California, November 5–9, 1995, pp. 391–400.
8. Goldfarb, C.F. The SGML Handbook. Claredon Press, Oxford, 1990

9. Grønbæk, K., Hem, J.A., Madsen, O.L., and Sloth, L. Cooperative Hypermedia Systems: A Dexter-based Architecture. Communications of the ACM, Vol. 37, No. 2, February 1994, pp. 65–74
10. Haake, A. and Haake, J.M. Take CoVer: Exploiting version support in cooperative systems. Proceedings of the InterCHI'93, Amsterdam, Netherlands, April 26–29, 1993, pp. 406–413
11. Haake, J.M., Bapat, A., and Knopik, T. Using a Hypermedia System for Systems Engineering. Proceedings of the East–West International Conference on Multimedia, Hypermedia, and Virtual Reality (MHVR'94), Moscow, Russia, September 14–16, 1994, pp. 63–68
12. Halasz, F.G. and Schwartz, M. The Dexter Hypertext Reference Model. Communications of the ACM, Vol. 37, No. 2, February 1994, pp. 30–39
13. Klas, W., Aberer, K., and Neuhold, E.J. Object-Oriented Modelling for Hypermedia Systems Using the VODAK Modelling Language. In: Dogac, A., Özsu, T., and Biliris, A. (Eds.): Advances in Object-Oriented Database Systems, NATO ASI Series F, Springer Verlag Berlin, 1994, pp. 389–433.
14. Lange, D.B. Object-Oriented Hypermodeling of Hypertext Supported Information Systems. Proceedings of the 26th Hawaii International Conference on System Sciences, Vol. 3, 1993, pp. 380–389.
15. Leggett, J.J. and Schnase, J.L. Viewing Dexter With Open Eyes. Communications of the ACM, Vol. 37, No. 2, February 1994, pp. 76–86
16. Lux, G. MUSE — A Technical Systems Engineering Environment. Proceedings ESS '93 European Simulations Symposium, Delft, Netherlands, October 25–28, 1993, pp. 293–298
17. Marshall, C.C., Halasz, F.G., Rogers, R.A., and Janssen Jr., W.C. Aquanet: A Hypertext Tool to Hold Your Knowledge in Place. Proceedings of the 3rd ACM Conference on Hypertext (Hypertext '91), San Antonio, TX, December 15–18, 1991, pp. 261–275
18. Mattos, N.M. Abstraction Concepts: The Basis for Data and Knowledge Modelling. Proceedings of the 7th International Conference on Entity-Relationship Approach, Rome, Italy, 1988, pp. 331–350.
19. Muth, P., Rakow, T.C., Weikum, G., Brössler, P. and Hasse, C. Semantic Concurrency Control in Object-Oriented Database Systems. Proceedings of the 9th IEEE Conference of Data Engineering (ICDE '93), Vienna , Austria, 1993, pp. 232–242.
20. Nanard, J. and Nanard, M. Using Structured Types to Incorporate Knowledge in Hypertext. Proceedings of the 3rd ACM Conference on Hypertext (Hypertext '91), San Antonio, TX, December 15–18, 1991, pp. 329–343
21. Rusinkiewicz, M., Klas, W., Tesch, T., Wäsch, J., and Muth, P. Towards a Cooperative Transaction Model – The Cooperative Activity Model. Proceedings of the 21th International Conference on Very Large Data Bases (VLDB '95), Zurich, Switzerland, September 11–15, 1995, pp. 194–205
22. Schütt, H.A. and Haake, J.M. Server Support for Cooperative Hypermedia Systems. Hypermedia – Proceedings der Internationalen Hypermedia '93 Konferenz, Zurich, Switzerland, March 2–3, 1993, pp. 45–56
23. Schütt, H.A. and Streitz, N.A. HyperBase: A Hypermedia Engine Based on a Relational Database Management System. Proceedings of the European Conference on Hypertext (ECHT '90), Versaille, France, November 1990, pp. 95–108
24. Streitz, N.A., Haake, J.M., Hannemann, J., Lemke, A., Schuler, W., Schütt, H.A., and Thüring, M. SEPIA: A cooperative hypermedia authoring environment. Proceedings of the 4th ACM European Conference on Hypertext (ECHT'92), Milan, Italy, November 30–December 4, 1992, pp. 11–22.
25. Streitz, N.A., Geißler, J., Haake, J.M., and Hol, J. DOLPHIN: Integrated meeting support across LiveBoards, local and remote desktop environments. Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'94), Chapel Hill, N.C., October 22–26, 1994, pp. 345–358
26. Tesch, T. and Wäsch, J. Transaction Support for Cooperative Hypermedia Document Authoring – A Study on Requirements. Proceedings of the 8th EDRG Workshop on Database Issues and Infrastructure in Cooperative Information Systems (EDRG-8), Trondheim, Norway, August 23–25, 1995, pp. 27–38
27. VODAK Manual Release 4.0. Technical Report, Arbeitspapiere der GMD No. 910, GMD, Germany, April 1995
28. Wäsch, J. and Aberer, K. Flexible Design and Efficient Implementation of a Hypermedia Document Database System by Tailoring Semantic Relationships. Proceedings of the Sixth IFIP Conference on Data Semantics (DS-6), Stone Mountain, Georgia, May 30–June 2, 1995.
29. Wäsch, J. and Klas, W. History Merging as a Mechanism for Concurrency Control in Cooperative Environments. To appear in Proceedings of the 6th International Workshop on Research Issues in Data Engineering: Interoperability of Nontraditional Database Systems (RIDE-NDS '96), New Orleans, Louisiana, February 26–27, 1996.
30. Wiil, U.K. and Leggett, J.J. Hyperform: An Extensible Hyperbase Management System. Technical Report, TAMU-HRL-92-003, Texas A&M University, July, 1992
31. Wiil, U.K. and Leggett, J.J. Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems. Proceedings of the ACM Conference on Hypertext (ECHT '92), Milano, Italy, 1992, pp. 251–261
32. Wiil, U.K. and Østerbye, K. Experiences with HyperBase – A multi-user back-end for hypertext applications with emphasis on collaboration support. Technical Report R-90-38, CS Dept., University of Aalborg, Denmark, October 1990