Hyperform: A Hypermedia System Development Environment

UFFE K. WIIL
Aalborg University
and
JOHN J. LEGGETT
Texas A&M University

Development of hypermedia systems is a complex matter. The current trend toward open, extensible, and distributed multiuser hypermedia systems adds additional complexity to the development process. As a means of reducing this complexity, there has been an increasing interest in hyperbase management systems that allow hypermedia system developers to abstract from the intricacies and complexity of the hyperbase layer and fully attend to application and user interface issues. Design, development, and deployment experiences of a dynamic, open, and distributed multiuser hypermedia system development environment called Hyperform is presented. Hyperform is based on the concepts of extensibility, tailorability, and rapid prototyping of hypermedia system services. Open, extensible hyperbase management systems permit hypermedia system developers to tailor hypermedia functionality for specific applications and to serve as a platform for research. The Hyperform development environment is comprised of multiple instances of four component types: (1) a hyperbase management system server, (2) a tool integrator, (3) editors, and (4) participating tools. Hyperform has been deployed in Unix environments, and experiments have shown that Hyperform greatly reduces the effort required to provide customized hyperbase management system support for distributed multiuser hypermedia systems.

Categories and Subject Descriptors: H.1.1 [**Models and Principles**]: Systems and Information Theory—*general systems theory;* H.2.1 [**Database Management**]: Logical Design—*data models;* H.2.4 [**Database Management**]: Systems—*distributed systems;* H.2.8 [**Database Management**]: Database Applications; H.3.4 [**Information Storage and Retrieval**]: Systems and Software; H.3.m [**Information Storage and Retrieval**]: Miscellaneous—*hypermedia*

General Terms: Design, Experimentation, Management

Additional Key Words and Phrases: Advanced hypermedia system architecture, extensible hyperbase management system, object-oriented extension language

This research was supported in part by the Danish Natural Science Research Council through Program 11-0061, the Faculty of Technical and Natural Sciences, Aalborg University, Denmark, through grant 6412-8, and the Texas Advanced Research Program under grant 999903-230.

Authors' addresses: U. K. Wiil, Department of Computer Science, Aalborg University, Fr. Bajers Vej 7E, 9220 Aalborg Ø, Denmark; email: kock@cs.auc.dk; J. J. Leggett, Department of Computer Science, Texas A&M University, College Station, TX 77843; email: leggett@cs.tamu.edu.

1. INTRODUCTION

Development of hypermedia systems[1] is a complex matter. The current trend toward open, extensible, and distributed multiuser hypermedia systems adds additional complexity to the development process. As a means of reducing this complexity, there has been an increasing interest in hyperbase management systems (HBMSs) [Campbell and Goodman 1988; Schnase 1992; Schütt and Haake 1993; Wiil and Leggett 1992; Zobel et al. 1991] that allow hypermedia system developers[2] to abstract from the intricacies and complexity of the hyperbase layer and fully attend to application and user interface issues.

Most current-generation HBMSs impose unnecessary complexity (or restrict desired flexibility) on the hypermedia system developer by providing fixed data models (hypermedia objects and operations) [Wiil and Leggett 1992]. Relying on fixed services in the hyperbase layer can force developers to make undesirable compromises in the design of the application and user interface layers [Akscyn et al. 1988; Wiil 1992]. Developers have to deal with the question "How do I make the best use of the services provided in the hyperbase layer?" In other words, the developer has to think in terms of the provided HBMS support when designing the other layers of the system. This often requires inelegant and inefficient workarounds in the development process to compensate for inadequate HBMS features.

This article presents the design, development, and deployment experiences of a dynamic, open, and distributed multiuser hypermedia system development environment called Hyperform. Hyperform is based on the concepts of extensibility, tailorability, and rapid prototyping of hypermedia system services. Hyperform provides a framework of general building blocks that can be extended and tailored (through object-oriented techniques) to match the specific HBMS needs of advanced hypermedia systems. By having extension facilities in the hyperbase layer, hypermedia system developers can avoid making undesirable design trade-offs due to fixed HBMS support and turn the above question into "Which services would I like the hyperbase layer to provide." Hyperform introduces extension facilities in both the hyperbase and application layers, enabling developers to optimally partition hypermedia functionality.[3]

[1]The term "hypermedia system" is used to describe a collection of tools (one or more) that include hypermedia functionality (anchoring and linking). A "tool" is any computer program/application that helps an end-user perform a specific task (e.g., text editors, drawing tools, spreadsheets, and mail tools).

[2]A "hypermedia system developer" is a person who constructs hypermedia systems to be used by end-users. Hypermedia systems basically can be developed in two distinct ways: by adding hypermedia functionality to existing (third-party) tools or by developing new hypermedia tools from scratch. Either way, tools must store and retrieve hypermedia data, and thus hypermedia system developers must deal with the complexity of the storage subsystem.

[3]In other words, Hyperform can assist hypermedia system developers in the process of constructing the hyperbase layer (of tools) and the parts of the application layer (of tools) that include hypermedia functionality.

Section 2 presents important data management issues that distinguish hypermedia systems from other information systems. We continue in Section 3 by presenting an overview of the Hyperform development environment. Section 4 presents a case study of the use of Hyperform in an (object-oriented) implementation of the Dexter Hypertext Reference Model [Halasz and Schwartz 1994]. In Section 5, we show how our work relates to other research in the hypermedia literature, and in Section 6, we discuss experiences and usability aspects of Hyperform. Section 7 concludes and provides an overview of future plans.

2. HYPERBASE RESEARCH ISSUES

Hypermedia systems pose several difficult data management problems. Most of these ultimately can be derived from the fact that hypermedia is a complex amalgam of information, structure, and behavior [Schnase et al. 1993b]. Data management facilities in hypermedia systems are usually found in the hyperbase layer of the architecture. Five broad categories of issues must be addressed in future HBMS research [Leggett et al. 1993; Schnase et al. 1993a]:

(1) *Models and Architectures.* Issues of scalability, extensibility, architectural openness, computation, interoperability, distribution, and platform heterogeneity are of critical importance [Leggett and Schnase 1994].

(2) *Node, Link, and Structure Management.* Data management facilities for hypermedia must address issues relating to object identity and naming, as well as constraints to ensure object and structure integrity. In addition, support for object composition, contexts, and views is critical [Halasz 1988]. The management of data types including spatial, temporal, image, sequence, graph, probabilistic, user defined, and dynamic is essential. The effective management of behavioral entities will be important in many settings.

(3) *Browsing/Query and Search.* Optimizing the synergy between hypermedia's navigational approach to data access and traditional query and search must be addressed at the hyperbase level. Important issues include the introduction of multilevel store indexing techniques, agency, hyperbase heterogeneity, extensibility, and optimization.

(4) *Version Control.* Effective support for versioning requires an understanding of precisely which entities need to be versioned and when version creation occurs. In hypermedia, there are opportunities to version structure as well as information. It is also important to understand how version control should be partitioned between the hyperbase and application levels [Hicks 1993].

(5) *Concurrency Control, Transaction Management, and Notification.* The types of interactivity and operations that characterize hypermedia systems create a requirement for managing short, long, and very long

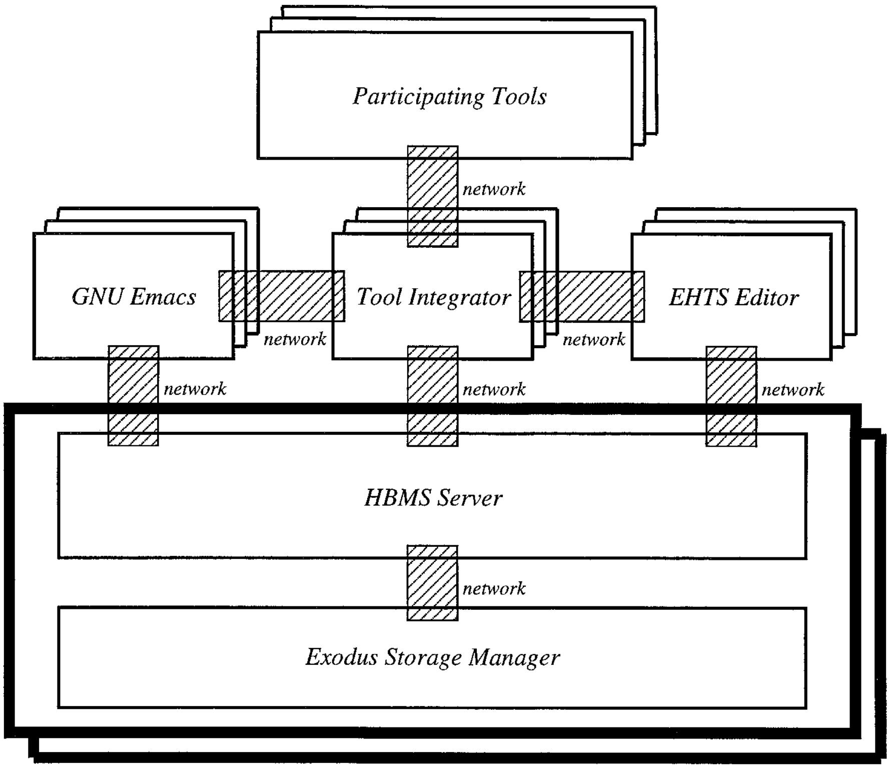

Fig. 1. Components of the Hyperform development environment. The tool integrator is an extensible and tailorable interface component between the HBMS server and the participating tools.

hyperbase transactions. HBMS support for collaboration and sharing of information is of critical importance [Wiil and Leggett 1993].

Although current-generation HBMSs address some of the above issues, research on HBMSs is still in the experimental phase. We have yet to see the first successful commercial HBMS addressing all of the above issues. Many ideas and technologies from the database, information retrieval, and information systems fields are being modified and reused in HBMS development.

3. THE HYPERFORM DEVELOPMENT ENVIRONMENT

Hyperform provides an open development environment that supports rapid prototyping of hypermedia data models, HBMSs, and hypermedia system architectures. The Hyperform development environment is comprised of multiple instances of four component types (see Figure 1): (1) the HBMS server, (2) the tool integrator, (3) editors (currently the EHTS Editor [Wiil 1991; 1992] and GNU Emacs [Stallman 1984]), and (4) participating tools.

Hyperform uses the Exodus Storage Manager [Carey et al. 1993] to manage physical storage.

The HBMS server is based on an internal computational engine that provides an object-oriented extension language which allows new data model objects and operations to be added at run-time. As discussed in the introduction, these capabilities give the hypermedia system developer the power, flexibility, and opportunity to decide where functionality should exist in the overall architecture, even to the point of modifying the underlying data model of the hyperbase. The HBMS server has a number of built-in classes (a library of reusable object-oriented software routines) that provide basic tool-independent HBMS features such as concurrency control, notification control (events), access control, version control, and query and search. These classes can be specialized using multiple inheritance to form virtually any type of HBMS support needed in hypermedia systems. It is possible to have more than one data model and HBMS configuration running in the HBMS server at the same time. We distinguish between a data model, which consists of a number of objects and their associated operations, and an HBMS configuration, which is a data model with policies and mechanisms for concurrency control, notification control, access control, version control, and query and search (or a subset of these features) built into the objects and operations. This allows heterogeneous hyperbases to coexist concurrently and makes Hyperform the first multihyperbase system.

Hyperform is based on an extended client-server architecture which supports the development of dynamic, open, and distributed systems (see Figure 1). The extended client-server architecture introduces a new component, the tool integrator. The tool integrator serves two major purposes. First, it supports the dynamic integration of (third-party) tools with the development environment. Second, it allows developers to experiment with different hypermedia system architectures such as centralized client-server architectures or distributed multihyperbase architectures.

During the development process, the provided editors allow multiple hypermedia system developers to collaborate on design, development, and deployment of HBMSs. The two integrated editors allow developers to retrieve built-in classes, edit class descriptions, and install new classes in the environment at run-time. By allowing developers to tailor and extend the built-in classes of the HBMS and tool integrator dynamically, the editors enable rapid prototyping of alternate hypermedia data models, HBMSs, and hypermedia system architectures. In the following, we give a detailed overview of the individual components of the Hyperform development environment.

3.1 The HBMS Server

Hyperform provides an HBMS server implemented around the Elk (Extension language kit) interpreter [Laumann 1993]. Elk is based on a Scheme dialect compatible (to a very high degree) with the Scheme standard

[Clinger and Rees 1992]. In addition, Elk contains many features that make it suitable as the basis for the HBMS server. Since Elk supports dynamic loading of object files, it is possible to extend Elk with features written in other languages. Elk can be compiled to run on a wide range of Unix architectures, and Hyperform is currently running on Sun Sparcstations.

The design of Hyperform attempts to address both efficiency and flexibility aspects of HBMS development. The performance-critical parts of the HBMS server (such as network and storage management) are written in an efficient language, while the remainder of the system (such as data-modeling facilities) has the flexibility of an interpreted language. The HBMS server has gone through several design-experiment iterations before reaching the present partition of functionality between efficiently coded libraries and Scheme-based functionality. The tendency in these iterations has been to move functionality from Scheme to the libraries to improve the performance of the HBMS server. Many of the specialized Scheme components have been replaced by calls to a set of general library operations without compromising the flexibility of the HBMS server. In this way, Scheme is used to glue general library operations together into more powerful HBMS server components. Example libraries built in this fashion include the Exodus Storage Manager Library and the Networking Library. Additional usage of Hyperform is likely to result in further optimization.

3.1.1 *Extending Elk into an Object-Oriented Database.* Hyperform achieves dynamic (run-time) extensibility at both the data model level and kernel level in the HBMS server by combining the metaclass concept with the idea that fundamental database features should be provided as classes in the HBMS server. The data model level provides data model objects and operations, while the kernel level provides fundamental database mechanisms such as concurrency control and version control. The metaclass concept allows hypermedia system developers to extend the HBMS server at the data model level, since developers can use objects (classes) to describe and extend the behavior of the hyperbase itself. By providing fundamental database features as built-in classes in the HBMS server, developers can tailor and extend the HBMS server at the kernel level as well as by specializing existing classes.

The heart of the HBMS server is an object-oriented data-modeling facility implemented in Scheme as an extension to the Scheme language. Basic object-oriented database (OODB) features such as object persistency, object identity, attribute and procedure encapsulation, object specialization by multiple inheritance, class (type) evolution by class versioning, and method invocation by message passing are supported. The HBMS server treats all instances and classes as objects and is self-contained, since all class descriptions (metadata) are saved as objects in the database.

The built-in classes of the HBMS server provide a general set of services without introducing special design policies. The classes specify a well-defined method interface to encapsulated data and behavior, enabling hypermedia system developers to abstract from basic implementation details. Developers can concentrate on tailoring and extending the provided services into an HBMS configuration that fulfills the chosen design policies.

The following sections explain the extension of the Elk Scheme interpreter into an OODB that has been specialized for hypermedia. We will deal with important object-oriented issues, such as the object concept, object persistency and identity, object specialization, attribute and procedure encapsulation, class evolution, and message passing. We will also touch upon some basic database mechanisms to deal with system matters, such as managing database access control. The purpose of the present work is not to invent new OODB technology or reinvent old OODB technology. Hyperform uses existing OODB techniques, where appropriate, in a straightforward manner. However, Hyperform combines different object-oriented techniques and database techniques in novel ways to address the difficult data management problems posed by hypermedia systems (see Section 2).

Object Concept. The object approach taken in the HBMS server is based on the fact that objects in hypermedia systems basically consist of a number of default attributes, dynamically allocated attributes, and a number of operations on these attributes. The basic object class of the HBMS server supports five categories of attributes: *read-only* (similar to "static" class variables), *read-write* (similar to instance variables), *write-once, calculated,* and *dynamic.* There is no restriction on the size and contents of attributes, and all attributes can be assigned an initial value in the class description. The value of read-only attributes remains the same in all instances of the class; read-write attributes can differ from instance to instance; write-once attributes can differ from instance to instance (but can only be written once); calculated attributes can contain any Scheme expression (which is evaluated when reading the attribute); and dynamic attributes can be added on-the-fly. Dynamic attributes can be read-write, write-once, or calculated. The former two categories of attributes have their origin in object-oriented programming languages and databases, while the latter three are added to deal with special requirements of hypermedia systems. For example, the presence of calculated attributes allows hypermedia system developers to address the concept of virtual structures, such as computed composites and computed links, in a very fundamental way [Halasz 1988]. Instead of storing the exact collection of components of a composite or the exact endpoint(s) of a link, the use of calculated attributes allows composite contents and link endpoint(s) to be stored as queries (Scheme expressions) that are evaluated at access time. The object class contains seven basic methods: three instance methods, get-instance, set-instance, and delete-instance, and four attribute methods, get-attribute, set-attribute, add-attribute, and delete-attribute.

Object Persistency, Identity, and Encapsulation. Every new object in the HBMS server is assigned a unique object identifier by the Exodus Storage Manager to ensure the uniqueness of objects in their entire lifecycle.

Objects are saved in Exodus every time they change to ensure data integrity. Every class in the HBMS server has its own environment in which the methods of the class are evaluated (and bound). Class environments are built when the HBMS server begins execution and when new classes are loaded into the HBMS server with the set-class method of Meta Class (see section *Class Evolution*). This solution enables fast method lookup and evaluation, since class descriptions are cached in the HBMS server and do not have to be retrieved from Exodus every time they are used.

Object Specialization. The Object class is the root of the class inheritance hierarchy. The object class can be specialized in two ways to define new object types in the HBMS server: (1) by adding new methods, which is similar to adding methods in a subclass, and (2) by adding default attributes. These attribute names will always be present in instances of that specific object type (subclass). Default attributes can be read-only, read-write, write-once, or calculated. A subclass inherits all methods and default attributes from its superclass(es) with the restriction that each method and attribute have a unique name. Notice that an attribute can be specified as *read-only* at one level and changed to *calculated* in a subclass.

Message Passing. Messages can be sent to objects in the following way:

```
(send object message . args)
```

The send mechanism performs method lookup and initiation. send can be used transparently over the network from the tool integrator and editors or from methods of the HBMS server classes.

Class Evolution. Meta Class provides basic operations on the class inheritance hierarchy: get-class, set-class, delete-class, and create-instance. These operations retrieve class descriptions, create new classes and new versions of existing classes, delete classes, and create instances of classes. The metaclass facility allows hypermedia system developers to extend the behavior of the HBMS server by implementing and refining a data model. Multiple versions of classes are maintained to allow classes, except the three basic classes (Meta Class, System Object, and Object), to evolve over time. The class-versioning facility allows developers to maintain multiple versions of data models to experiment with alternate data model designs and implementations. Since many of the fundamental database features are provided as built-in classes in the HBMS server, the metaclass and class-versioning concepts also allow developers to experiment with alternate HBMS configurations (e.g., policies and mechanisms for concurrency control).

A new (version of a) class is created by sending the set-class message to Meta Class containing a class description consisting of two parts: a list of default attributes and specifications of methods operating on the attributes. The following example shows a simplified description of the access control (AC) Object:

```
((class-name AC-object)
(write-once-att (owner ()) (created ()) )
(read-write-att (modified-by ()) (modified ()) (permission "gsdg__g__") (group ()) )
(super-class object)
(methods
  (define (AC-init self . args). . . )
  (define (change-permission self permission). . . )
  (define (change-group self group). . . )
  (define (get-permission self). . . )
  (define (set-permission self). . . )
  (define (delete-permission self). . . ) ))
```

The basic part of class descriptions must be written in the Scheme-based object-oriented extension language. Since the HBMS server allows dynamic loading of object code, parts of new classes (typically the body of methods) can be written in other languages for reasons of efficiency. In this way, method specifications in class descriptions are reduced to interfaces between the message-passing mechanism in the HBMS server and the functions provided in the object code.

Basic Database Mechanisms. The System Object manages access control at the database level by storing information on valid end-users, groups of end-users, and superusers (hypermedia system developers) and by providing methods to maintain this information. System information is kept persistently in an instance of System Object. The connect and disconnect methods are located here, as well as methods that deal with other system matters such as permissions to operate on the metadata (classes) in the HBMS server. By default Hyperform is an open development environment allowing all users (database administrator, hypermedia system developers, and end-users) access to all facilities. Therefore, when users are not distinguished as end-users and superusers, all users have superuser privileges. The database administrator can decide to limit the access to a small group of users by maintaining lists of valid end-users and superusers. End-users can only operate on the instances (data) in the HBMS server, while superusers are also allowed to work on classes (metadata) and to maintain information on end-user groups.

3.1.2 *Object Subclasses.* To help address the difficult data management problems posed by advanced hypermedia systems (see Section 2), we introduced five subclasses of Object to provide basic tool-independent HBMS support in the areas of concurrency control (CC Object), notification control (NC Object), access control (AC Object), version control (VC Object), and query and search (QS Object). Initially, the five subclasses of Object were developed from scratch using the object-oriented kernel (Meta Class and Object class) [Wiil and Leggett 1992]. After incorporation of the Exodus Storage Manager in Hyperform, the functionality of the concurrency control, version control, and query and search classes were modified to rely (to some extent) on the facilities of Exodus. Since Exodus does not provide support for notification control or access control, these classes remained unmodified.

To our knowledge, no OODB has libraries that are specialized to hypermedia without introducing specific policies. Therefore, Hyperform uses Exodus, rather than a full-fledged OODB, to provide efficient storage and retrieval of objects as well as low-level database transactions that ensure data integrity. In addition, Hyperform adds object-oriented data management facilities that have been designed to meet the specific requirements of hypermedia.

We want to emphasize that Hyperform does not introduce a specific data model. The development of the five subclasses of Object is an attempt to implement as much support for hypermedia systems as possible without introducing data-modeling policies. Ideally, these basic building blocks should not introduce design decisions or in any way restrain the data-modeling and HBMS configuration process. Reaching a level of support in the building blocks that is both powerful enough to be useful in advanced hypermedia systems and at the same time general enough to be used in different types of tools is indeed difficult. What is considered to be general hypermedia data management services to some types of tools can be too specific or too general to others. The development of the five subclasses of Object has gone through numerous iterations before reaching the present set of services.

The services of the five subclasses of Object have been kept general, based on the idea that future usage of Hyperform will build a large library of useful hypermedia data models and HBMS configurations. Currently, the library consists of a number of classes that implement different data models (e.g., the Dexter model [Halasz and Schwartz 1994]) and HBMS configurations (e.g., Aalborg University's HyperBase [Wiil 1993a]). There are two types of library classes: those that specialize the functionality of the five subclasses of Object and those that implement hypermedia data model components such as anchors, links, nodes, and composites. Library classes can be reused in other development scenarios, saving development time and adding to the size of the library. In the following, we will briefly describe the most important services of the five subclasses of Object.

Concurrency Control Object. Concurrency control is an important issue for database systems, and much work has been done in this area [Barghouti and Kaiser 1991; Bernstein and Goodman 1981], but the results cannot be directly adopted for HBMSs. HBMSs must provide special support for collaborative work [Halasz 1988; Wiil and Leggett 1993], requiring adjustments to normal database notions of concurrency control. The HBMS server provides enhanced support for collaborative use. CC Object provides short database transactions combined with user-controlled locking. User-controlled locks provide support for long-duration updates to objects in the database and are shared, fine grained (attribute-level), and persistent.

A (long) collaborative update session in Hyperform is initiated with an explicit lock request. Each operation in the session (such as frequent saves) is performed within short database transactions in the Exodus Storage Manager. Locked objects can be read by other users, and granted locks are stored persistently. In this way, Hyperform is able to recover from both client and server crashes within long collaborative sessions [Wiil and Leggett 1993].

Typically, transactions are used to group basic database methods together into more powerful atomic operations at the application level. In Hyperform, atomic operations involving more than one method are supported by moving the operation from the application level to the HBMS using the provided dynamic data model extension facilities. Hyperbase-level operations in Hyperform make use of the transaction mechanism in Exodus to ensure atomicity and recoverability. This technique speeds the operation, since internal database operations are much faster than application-level operations over the network.

Notification Control Object. Notification control allows end-users/tools to be notified of important actions on the shared network of nodes and links performed by other end-users/tools of the system [Halasz 1988]. A notification control mechanism is necessary for supporting collaboration [Wiil and Leggett 1993]. NC Object provides a powerful event notification mechanism that uses the expressive power of the Scheme language in event subscriptions. The event mechanism is asynchronous and fine grained (attribute level), and event subscriptions are kept persistently. Events can be any Scheme expression and have access to all variables, functions, and objects in the HBMS server. For example, the following simple expression

```
(event (lambda ()
  (if (and (equal? NC-attribute “data”)
        (equal? NC-operation “Write”))
    (list NC-entity user)
    #f)))
```

stipulates that all write operations performed on data attributes will cause an event specifying the target object and the end-user performing the operation. Event expressions are evaluated when the send-events method is invoked. All expressions evaluating to anything other than false cause an event to be generated and transmitted to the subscribing user. send-events can be included in all methods in subclasses created in the HBMS server and is (by default) invoked when using the methods of Meta Class and System Object, since these cannot be versioned or subclassed. send-events can also be invoked with specific parameters allowing “high-level” events to be generated (for instance, events describing actions performed in the tool integrator instead of the HBMS server).

The event mechanism provides many capabilities. The mechanism can be used to make the HBMS server an active hyperbase. Specific operations can trigger other operations which, for instance, could update variables and objects in the HBMS server. Since the event mechanism has access to all the computational power in Hyperform, it can be used to trigger arbitrarily complex internal computations such as creating virtual structures [Halasz 1988].

Access Control Object. As hypermedia systems evolve from single-user to multiuser systems, the need for controlling end-user access to shared objects arises. The simple read/write protection scheme is not appropriate and should be augmented with at least a third level of protection—annotate—allowing end-users to attach links to objects to which they have read (but not write) access [Catlin et al. 1989].

The design of AC Object general services was influenced by the protection mechanism in Unix. We support three different levels of object protection, *get, set,* and *delete* which correspond to the basic operations on instances and attributes, and we adopt the notion of "user/group/others" from Unix. Since there is no predefined notion of nodes, links, and composites in the HBMS server, it is not possible to talk about annotation rights as being part of the general services of AC Object. Basically, annotation rights can be implemented with *set* access. Once a data model is loaded into Hyperform, AC Object can easily be specialized to include an annotate access right (or other levels of protection).

Version Control Object. Versioning is an important feature for hypermedia systems [Halasz 1988]. Until recently, little research had been done to uncover appropriate versioning models for hypermedia [Halasz 1991]. The requirements of versioning in hypermedia systems can be categorized into two main areas: versioning *data* and versioning *structure* [Hicks 1993]. Versioning data has its parallel in version control, while versioning structure has its parallel in configuration management [Rohrbach and Seiwald 1988].

Versioning structure is heavily dependent on the hypermedia data model used in the system. It is impossible to provide general HBMS support for versioning structure in hypermedia without introducing a hypermedia data model. Versioning data is independent of the hypermedia data model, but still requires policies to be introduced. For instance, the versioning model used might be timeline, tree, network, or a new hypermedia versioning model. VC Object was created to address data-versioning issues. Since structure is contained in the attributes of data model objects in the HBMS server, it is likely that structure versioning would be done by subclassing VC Object (or one of its variants) once a particular data model is introduced.

The data-versioning support in VC Object was inspired by RCS [Tichy 1985] and Gypsy [Cohen et al. 1988] and is based on the tree model. VC Object builds on the simple support for multiple versions of objects in the Exodus Storage Manager. VC Object supports revisions, variants, and releases of objects in delta or fully constituted form. It is possible to check-out instances of objects for update and later check them in as new revisions, variants, or releases of the instance. VC Object manages version numbering based on the type of instance versioning requested at check-out time. At check-in time, one can specify whether the previous version of the object should be stored as a delta or as a complete copy. At any time a message can be sent to objects that changes the method of storage. Versions of an instance are kept in version groups, and earlier versions can be retrieved through the query mechanism. We use a specialized form of backward delta storage (geared toward the storage method in the HBMS server) combined with the object version facilities of the Exodus Storage Manager to reduce the amount of storage required and to allow flexible access to previous versions of the instance.

A small number of researchers are currently addressing hypermedia versioning issues [Haake 1992; Hicks 1993; Østerbye 1992]. Hyperform, due to its extensible nature, can be used as a research vehicle for experimenting with different versioning models for hypermedia. In particular, Hyperform's rapid prototyping facility can be used for addressing the issue of partitioning version support among the various architectural layers (e.g., hyperbase versus application) [Hicks 1993].

Query and Search Object. The basic information access metaphor in hypermedia is navigation. In general, navigation becomes inefficient as the hypermedia information space grows larger and larger. Techniques from database and information retrieval (IR) systems should be incorporated into hyperbase technology to provide alternate ways of accessing information in hypermedia. Halasz [1988] suggests query-based mechanisms supporting both *content* and *structure* search. Content search involves IR techniques such as keyword, index, and full-text retrieval and database techniques such as query-based retrieval. Structure search involves a special structural query language capable of retrieving specified subgraphs of the network and therefore depends on the data model in the system.

We provide basic content search facilities in the HBMS server that can be extended to provide support for more powerful IR methods. The content search mechanism of QS Object is list-based and supports the use of all list-manipulating functions of the Scheme language. In addition, we provide a general filtering function capable of matching specific values in attributes (remember, all information in the HBMS server is stored in attributes) and basic list operations such as union, intersect, and subtract. We can also access all internal indexes of the HBMS server, retaining information on classes, instances, locks, events, etc. These basic methods (inspired by Fuller et al. [1991]) can be composed into very powerful content searches using the composing mechanisms of the Scheme language (control structures such as IF, COND, and CASE and operators such as AND, OR, and NOT).

QS Object builds on the Exodus Storage Manager support for indexes (B^+ trees and linear hashing). Exodus indexes are used to maintain internal HBMS server information. Due to the update overhead of indexes, we only maintain information that is frequently accessed (e.g., the coherence between classes and their instances). Hypermedia system developers can make use of the indexing capabilities of the Exodus Library when designing data models and HBMS configurations.

We do not provide direct support for structure search, since we do not want to introduce a fixed data model. Although, since all structural information is stored in attributes in the HBMS server, structure search mechanisms can be built on top of the basic content search facilities once a data model is introduced.

3.1.3 *Attribute Cache and Libraries.* First experiences with Hyperform indicated that large attributes were not handled very efficiently in the Elk Scheme interpreter. To improve the performance of the HBMS server, we have created a special cache to handle large attributes in objects. When a message arrives, the network library interface moves large attributes from the message into the cache and replaces the attributes in the message with cache identifiers. Before the object is forwarded to the storage manager library, cache identifiers are replaced by the actual data in the storage manager interface. In this way large data objects are transparently handled in efficiently coded libraries all the way from the network to the storage manager and vice versa. We avoid spending valuable time converting large attributes to and from Elk Scheme format. Only in special cases, such as reading (evaluating) calculated attributes, is it necessary to transform such attributes into Elk Scheme format.

3.2 The Tool Integrator

The tool integrator (TI) is an extensible, tailorable interface between the HBMS server and participating tools. The TI is also based on Elk, giving it extension features similar to those of the HBMS server. Since Elk can be extended on-the-fly, the TI supports dynamic integration of new tools into Hyperform making the architecture dynamic, open, and distributed. The TI has object-oriented data-modeling capabilities similar to those of the HBMS server. In fact, the data-modeling component in the TI is a modified version of the object-oriented kernel (Meta Class and Object class) in the HBMS server. The main difference is that the TI uses the HBMS server Object class to provide persistent storage for instances and to store its class descriptions. When the TI is invoked, relevant class descriptions are retrieved from the HBMS server and installed (evaluated).

The extended client-server architecture supports extensibility at two levels in systems: hyperbase and application, leaving it up to the hypermedia system developer to decide in each case where extensions give best results in terms of flexibility and efficiency. In some cases it might be best to place the extensions in the HBMS server (if they require access to many objects) and in other cases in the TI (to save network communication). More specific features can be put into the HBMS server than the TI, since it is easier to determine the necessary support at the hyperbase level than at the application level. The support at the hyperbase level should be tool independent while the support at the application level should focus on providing features that are common for participating tools. Since we cannot foresee the nature of participating tools, common features must be implemented by the hypermedia system developer using the extension (object-oriented data-modeling) facilities of the TI.

The TI supports integration of both external and internal tools. External tools can be integrated via the Unix and network libraries. Internal tools can be integrated by loading them into Elk via the dynamic object file-loading facility. The Scheme extension language is used to facilitate communication among different tools and between tools and the HBMS server. It may be difficult to integrate independently written tools, since they have been developed as standalone programs. Noninteractive tools can be invoked via the Unix library, and interactive tools can be extended to communicate with the TI via the network library.

We can distribute the HBMS server across several machines in the network. The architecture has also been designed with multiple HBMS servers in mind. The TI can be extended to interact with multiple HBMS servers or other storage devices (e.g., CD-ROM and optical disc) and data repositories (e.g., file systems and databases). In this way, the HBMS servers will not have to change to provide a truly distributed heterogeneous HBMS incorporating several different data repositories. The TI allows hypermedia system developers to experiment with different hypermedia system architectures including centralized, client-server architectures and distributed, multihyperbase architectures.

An additional advantage of the TI is that it can maintain a shared representation (cache) of important hyperbase data and structure (based on events from the HBMS server) used by different tools. Different views (editor, graphical browser, other browsers, etc.) on the data and structures can be based on the common cache which speeds operations in the tools, saves internal memory and network communication, and reduces the server load in event-driven approaches to data distribution [Wiil 1992]. Another example of specific TI functionality is protocol transformation. Independently written tools may use a storage (network) protocol different from that of the HBMS server. The TI can be used as an intermediate component transforming to and from the HBMS server protocol. In other words, the TI can be used to introduce storage (network) transparency similar to Unix Network File Systems (NFS).

3.3 The Editors

The Hyperform development environment supports the use of two editors for customization of the provided HBMS services into new data models and HBMS designs: GNU Emacs [Stallman 1984], the well-known extensible editor and, EHTS Editor [Wiil 1991; 1992], a multiuser hypermedia text editor. These two editors represent two different development styles available in the Hyperform development environment.

3.3.1 *GNU Emacs.* GNU Emacs [Stallman 1984] was the first editor interfaced to the Hyperform development environment. The send mechanism can be used transparently from Emacs over the network to the HBMS server (either directly over the network or through a tool integrator). In this way hypermedia system developers can use the powerful editing facilities of Emacs when developing class descriptions for Hyperform (HBMS server or tool integrator). When a class description is ready to submit to Hyperform (HBMS server or tool integrator) as a new (version of a) class, the send mechanism is used to communicate the message and install the new class. Thereafter, the send abstraction can be used to test the facilities of the new class (version) from within Emacs. Emacs uses the Unix file system to store Hyperform class descriptions. When the EHTS editor was ported from Aalborg University's HyperBase [Wiil 1993a] to Hyperform, Emacs was used to develop and test the class definitions used to implement the HyperBase data model in the HBMS server. This experiment is described in detail in Wiil [1993b].

3.3.2 *EHTS Editor.* The EHTS Editor [Wiil 1991; 1992] is based on Epoch [Kaplan et al. 1992] (an extension to GNU Emacs). The EHTS Editor provides two interesting extensions to the Emacs-based development style in the Hyperform development environment.

Hypermedia System Developers Can Collaborate in the Design Process. The EHTS Editor enables a group to collaborate on a shared task. Changes made on shared data by one developer are immediately visible to all other members of the group. Group members can communicate in real-time and can send asynchronous messages within the editor, thus enabling collaboration among members separated by time as well as space. The EHTS Editor provides contention resolution at the level of attributes in nodes and links and allows any number of users to simultaneously read and display the data field of a given node in a window on the screen. The EHTS Editor allows locked objects to be read by any number of users; however, permission to make modifications to the data field is restricted to one user at a time. Locks are allocated when the user invokes the editor lock command (e.g., by simply typing on the keyboard) indicating a change in mode from browse to edit. Locks are deallocated when either the editor unlock command is invoked or when the window is closed. All readers are notified as soon as possible that a data field they are accessing may be changed. Readers are provided with four types of modification notices:

(1) *Intention.* All readers are notified when one person signals intention to modify the data field of the node by obtaining a lock. The readers also get the name of the person, enabling contact through use of the internal talk mechanism (real-time communication). Readers can then subscribe to the event corresponding to when the writer unlocks the data field.
(2) *Update.* When the writer actually writes the modified data field of the node onto the shared database, all readers of the data field automatically get the contents in the data field display updated with modifications made by the writer (real-time monitoring).

(3) *Completion.* When the writer is finished modifying the data field of a node, users having subscribed to this event get notified that the data field of the node has been unlocked and is write accessible.
(4) *Deletion.* When a node is deleted, the display of the node is removed from the screens of all readers.

Hypermedia System Developers Can Link Together Related Design Material. In contrast to Emacs, the EHTS Editor is an integrated part of the Hyperform development environment and uses the data storage and data-sharing capabilities of Hyperform to store class descriptions. In the EHTS Editor, hypermedia system developers can link together different versions of class descriptions, as well as test data and documentation of classes to the class descriptions. Class descriptions, test data, and documentation originating from a specific development process can therefore be stored in Hyperform and made accessible as a hypermedia network interconnecting all relevant documentation inside Hyperform.

It is possible to customize and extend all parts of the EHTS Editor. The Hyperform development environment provides both the source code implementing the EHTS Editor and the source code implementing the HyperBase data model in the HBMS server.

4. A CASE STUDY IN HBMS DEVELOPMENT

Development of an HBMS in Hyperform typically consists of the following five steps:

(1) *Analyzing the Problem Domain.* The problem could be to provide hypermedia services for a particular hypermedia system (a set of hypermedia tools), to provide hypermedia capabilities (anchoring and linking) to a set of nonhypermedia tools, or to simulate the functionality of an existing hypermedia data model or HBMS. This step typically results in a list of requirements that the HBMS must fulfill.
(2) *Designing the Hypermedia Services.* This step involves mapping the list of requirements to a set of data model objects and operations fulfilling the requirements. In this step the hypermedia system developer must determine to what extent the built-in classes can be used in the implementation of the hypermedia services and what services need to be developed from scratch. This step typically results in a list of data model objects, a sketch of how they relate to each other in terms of inheritance, and a specification of operations available on the individual objects. Hyperform allows the hypermedia system developer to partition the functionality of the hypermedia data model between the hyperbase layer (the HBMS server) and the application layer (the tool integrator). A layered design will result in two design specifications, one for the HBMS server and one for the tool integrator.
(3) *Writing the Class Descriptions.* In this step the hypermedia system developer writes the descriptions of the specified classes in the Scheme-

based object-oriented development language. The developer can decide to use either of the provided editors in the coding process.

(4) *Installing the Class Descriptions.* When the class descriptions have been fully or partially implemented, the hypermedia system developer can install them in either the HBMS server or the tool integrator by sending a message to the Meta Class of the particular component. The developer can decide to fully implement the classes before installing (and testing) them or to develop the classes in a number of iterations adding on more and more of the specified operations.

(5) *Testing the Class Descriptions.* Before the hypermedia data model is used with the desired tools, the functionality of the classes may be tested by sending messages from the editors to the classes. The final test is, of course, when the tools start to use the developed hypermedia services.

When the hypermedia data model is operational, the developer can refine the model by repeating as many of the above steps as desirable. Each step will create a new version of the hypermedia data model in Hyperform. Reasons to refine the data model could be to include additional functionality not anticipated at the time of development or to improve the performance of the data model by optimizing certain time-consuming operations. As mentioned, bodies of operations can be implemented in an efficient (compiler-based) language and interfaced in the class descriptions to improve the overall performance.

We now turn our attention to a case study that demonstrates the use of Hyperform in a data model and HBMS development process. We have chosen to base the case study on the most influential hypermedia reference model in the literature—the Dexter Hypertext Reference Model [Halasz and Schwartz 1994].

4.1 The Dexter Hypertext Reference Model

The Dexter model was developed by a group of leading hypermedia researchers in a series of workshops from 1988 to 1990. The Dexter model is an attempt to capture some of the best design ideas from that time's most prominent hypermedia systems. The goal of the Dexter model is to provide a basis for comparing systems as well as for developing interchange and interoperability standards [Halasz and Schwartz 1994].

The Dexter model divides hypermedia systems into three layers having well-defined interfaces as illustrated in Figure 2. The *runtime* layer describes the mechanisms supporting the end-user's interaction with the hypertext. The *storage* layer describes the network of nodes and links that is the essence of hypermedia. The *within-component* layer covers the content and structures within hypermedia nodes. The focus of the model is on the storage layer as well as the mechanisms of *anchoring* and *presentation specification* that forms the interfaces between the storage layer and the within-component and runtime layers, respectively.

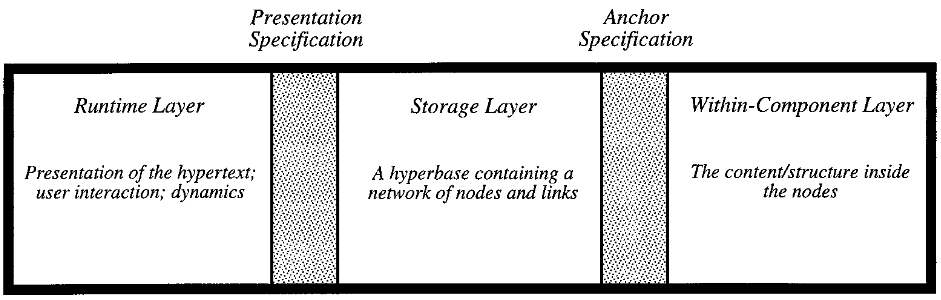
Presentation Specification
Anchor Specification
Runtime Layer
Presentation of the hypertext; user interaction; dynamics
Storage Layer
A hyperbase containing a network of nodes and links
Within-Component Layer
The content/structure inside the nodes

Fig. 2. An overview of the Dexter Hypertext Reference Model layers and interfaces.

The storage layer provides a basic, persistent object type, *component.* Components are generic containers of data and consist of two parts: *component information* and *component contents.* Component information is comprised of a set of *attributes,* a *presentation specification,* and a set of *anchors. Component contents* are the actual data stored in the component. Every component has a globally unique identity which is captured by its unique identifier. The Dexter model specifies three component subtypes: *atomic, link,* and *composite.* The atomic component type is an abstraction over the node concept. Link components include a list of *specifiers,* each consisting of a presentation specification, a direction (from, to, bidirect, or none), and component and anchor identifiers. Composite components provide a hierarchical structuring mechanism. A *hypertext* is a network composed of hierarchies of data-containing components which are interconnected by relational links.

The within-component layer is purposefully not elaborated within the Dexter model. No attempts are made to provide interpretation of component contents and internal structure. The tools using the HBMS are responsible for maintaining internal component data such as content selections for link anchoring.

The interface between the storage and within-component layers is based on the notion of *anchors.* Anchors provide a mechanism for addressing locations or items within the content of an individual component. Anchors consist of an *identifier* that can be referred to by links and a *value* that picks out the anchored part of the material.

The runtime layer is responsible for handling components, anchors, and specifiers at runtime, thus capturing the dynamic and interactional aspects of the hypermedia system. The runtime layer supports *sessions* that manage interaction with particular *hypertexts* and *instantiations* that manage interaction with particular *components.* The runtime layer provides user interface facilities through basic functionality for accessing, viewing, and manipulating hypermedia networks.

The interface between the runtime and the storage layers is based on the notion of *presentation specifications* which determine how components are presented at runtime. Presentation specifications might be dependent on the presentation tool, the component itself, or on the access path to the

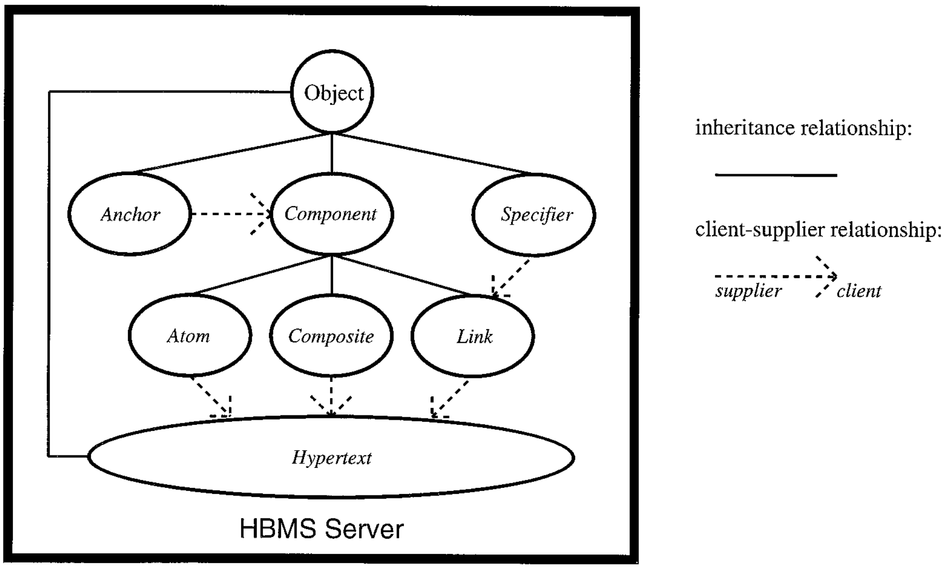

Fig. 3. Dexter storage layer classes in the HBMS server. All classes are subclasses of the persistent "Object" class. The "Component" class is a client of the "Anchor" class. The "Link" class is a client of the "Specifier" class, and the "Hypertext" class is a client of the "Atom," the "Composite," and the "Link" classes.

component (on the link) and might include information on screen location, size of the presentation window, or presentation modes (such as read-only or edit).

4.2 Implementing the Dexter Model in Hyperform

This section presents a brief overview of the object-oriented implementation of the Dexter model in Hyperform. We have only implemented the core model to avoid unnecessary interpretations of the semantics of the model [Leggett and Schnase 1994]. Further elaboration of the concepts and semantics of the core model pertain to a particular hypermedia system.

The layered HBMS approach in Hyperform provides a perfect foundation for the Dexter model approach to hypermedia systems. The within-component layer corresponds to the Exodus Storage Manager; the Dexter storage layer corresponds to the HBMS server; and the runtime layer corresponds to the tool integrator in Hyperform.

The Exodus Storage Manager provides efficient support for storage of different media types. It is the responsibility of participating tools to maintain internal component data stored in the Exodus Storage Manager.

Dexter storage layer concepts are implemented in the HBMS server using the object-oriented data-modeling facility (see Figure 3). We have separated anchor and specifier information from components; therefore, component subtypes store anchor IDs instead of the actual anchor data, and links store specifier IDs instead of the actual specifier data. All classes

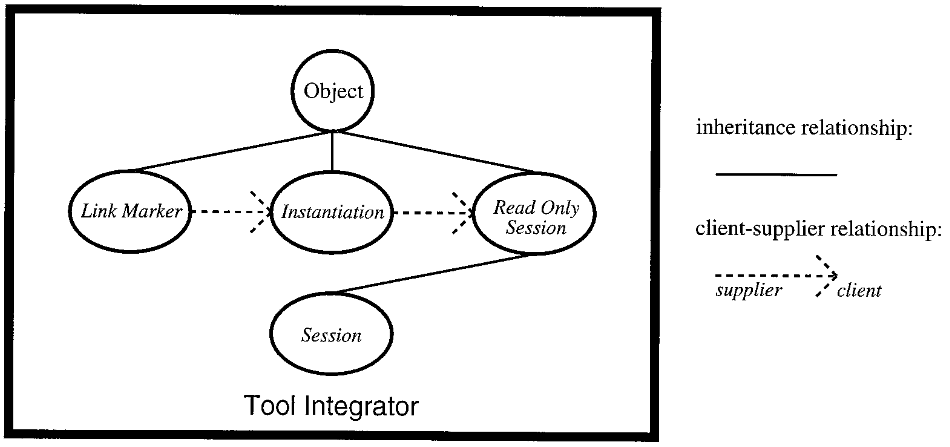

Fig. 4. Dexter runtime layer classes in the tool integrator. All classes are subclasses of the nonpersistent "Object" class. The "Read-Only Session" class is a client of the "Instantiation" class, which in turn is a client of the "Link Marker" class.

are persistent, since they are subclasses of the persistent object class. The "Hypertext" class conforms to the Dexter model concept of a hypertext.

Dexter runtime layer facilities are implemented in the tool integrator (see Figure 4). We have separated link markers from instantiations which implies that instantiations maintain a list of link marker IDs. Tool integrator classes are nonpersistent. This matches the requirements of the Dexter model: runtime layer classes use the facilities of the storage layer to obtain persistency. The "Session" and "Read-Only Session" classes conform to the Dexter model runtime specifications. A read-only session is a session that keeps no history and does not support create, save, and delete operations on the hypertext.

Typically, a hypermedia system will require several extensions to the core model, such as

(1) dealing with multiuser, collaboration, and versioning issues,
(2) dealing with different data types, different semantic relations between data types, and different aggregation mechanisms, and
(3) dealing with issues of openness and integration of different independently written (third-party) tools into the hypermedia system framework.

Elaborations and extensions to the core model can easily be performed in Hyperform, based on the current Dexter model implementation. Addressing the first set of issues would require using the built-in services for concurrency control, notification control, version control, access control, and query and search to enhance the facilities of the anchor and component classes in the HBMS server. Addressing the second set of extensions would require adding anchor, atom, link, and composite subtypes to provide a richer set of hypermedia data model objects. Anchor, atom, link, and composite subtypes can be added to the Dexter model storage layer by creating new classes in the HBMS server inheriting from the core Dexter storage layer classes shown in Figure 3. Aspects of openness and integration of third-party tools can be achieved by extending the runtime layer in the tool integrator to handle communication with different types of tools. This would be accomplished by adding new classes that specialize the behavior of the core Dexter runtime layer.

4.3 Important Results

Hyperform is well suited for implementing the Dexter model. The layered approach to HBMSs in Hyperform with object-oriented data-modeling facilities in both the HBMS server and the tool integrator constitutes a very powerful development environment. Hyperform supports partitioning of hyperbase (data model) functionality between the hyperbase and application layers—matching the partitioning of functionality between the storage and runtime layer in the Dexter model.

During the experiment, GNU Emacs was used to develop and test class descriptions for both the HBMS server and the tool integrator. Classes were stored in the Unix file system and sent to either the tool integrator or the HBMS server (via the tool integrator) whenever they were ready for testing. Tests were then performed by invoking methods from within Emacs.

The case study made the rapid prototyping facilities of Hyperform evident. The Dexter model was developed in less than two weeks by an experienced Hyperform developer. This effort included analysis of the Dexter model [Halasz and Schwartz 1994], design, implementation, and test. Less experienced Hyperform developers would, of course, have to put more effort into the design and implementation phases. The Dexter model classes each consist of 10 to 100 lines of Scheme code, for a total of 22KB of code in the entire experiment (13KB to implement the storage layer in the HBMS server and 9KB to implement the runtime layer in the tool integrator).

5. COMPARISON WITH RELATED WORK

Recently there has been an increasing interest in developing formal models for hypermedia. Among the most notable are those presented in Afrati and Koutras [1990], Delisle and Schwartz [1986], Furuta and Stotts [1990], Garg [1988], Halasz and Schwartz [1994], Lange [1990], Schnase et al. [1993a], Schütt and Streitz [1990], and Tompa [1989].

Many hypermedia models have been implemented in HBMSs. Prominent HBMS projects include work on the Hypertext Abstract Machine (HAM) [Campbell and Goodman 1988], GMD-IPSI's HyperBase [Schütt and Streitz 1990] and Cooperative Hypermedia Server (CHS) [Schütt and Haake 1993], University of North Carolina's Distributed Graph Server (DGS) [Shackelford et al. 1993], Texas A&M University's HB1 [Schnase et al. 1993a; 1993b], HB2 [Schnase 1992], and HB3 [Leggett and Schnase 1994], RMIT's

Hyperion [Zobel et al. 1991], and Aalborg University's HyperBase [Wiil 1993a]. The Dexter model has previously been implemented as part of the DeVise Hypermedia project at Aarhus University [Grønbæk and Trigg 1992; Grønbæk et al. 1993].

Unlike the above HBMS approaches, Hyperform implements no specific data model. Instead, we provide a rich set of basic tool-independent HBMS features that can be specialized to match the specific needs of advanced hypermedia systems. Hyperform is unique in its support for development and experimentation of hypermedia data models, HBMS configurations, and hypermedia system architectures through a rapid prototyping approach. Hyperform can be used to implement a wide variety of existing formal models and simulate existing HBMSs. Currently, we have an implementation of Aalborg University's HyperBase and the Dexter model in Hyperform.

Hyperform promotes an open hypermedia architecture allowing existing and future tools to be integrated under a common information system model. Other approaches to open hypermedia architectures and systems include Sun's Link Server [Pearl 1989], Proxhy [Kacmar and Leggett 1991], HB1 [Schnase et al. 1993a; 1993b], HB2 [Schnase 1992], HB3 [Leggett and Schnase 1994], Multicard [Rizk and Sauter 1992], DHM [Grønbæk and Trigg 1992; Grønbæk et al. 1993], Microcosm [Davis et al. 1992], DHT [Noll and Scacchi 1991], and ABC [Smith and Smith 1991].

6. DISCUSSION

This section takes a closer look at some of the features that make the Hyperform approach distinctive. We describe our experiences and discuss usability aspects of Hyperform.

6.1 Experiences

In addition to the development of the Dexter model and simulation of Aalborg University's HyperBase, Hyperform has been used in several development projects both inside and outside Aalborg University. The experiences from these experiments have some common characteristics.

Most important, experiments have supported the claim that the Hyperform development environment greatly reduces the effort required to develop customized HBMS support for distributed hypermedia systems. The rapid prototyping facilities of Hyperform makes experimentation with new hypermedia data models and HBMS configurations a matter of weeks of effort rather than months or years. The development of Aalborg University's HyperBase showed the time it takes to simulate a HAM generation HBMS in Hyperform is on the order of weeks compared to the two man-years it took to develop HyperBase. The Hyperform simulation is performed by 13KB of Scheme code compared to 350–400 KB of HyperBase C++ code [Wiil and Leggett 1992]. The Dexter, HyperBase, and other experiments show that the functionality of the built-in classes are at a level of generality that allows different hypermedia models to be developed. At the same time the built-in services provide a library of hypermedia data management functions that increase the effectiveness in experimentation and development of hypermedia models. The built-in services make Hyperform an effective testbed for research on the five categories of HBMS issues mentioned in Section 2.

Hyperform allows developers to partition functionality between the hyperbase and the application level, based on flexibility and efficiency considerations. The provided extension facilities at both the hyperbase and the application level can be used to experimentally determine the optimal partitioning of functionality. For example, in the HyperBase simulation experiment, we used this flexibility to gain efficiency by moving complex operations (e.g., operations that require a set of objects to be identified and then certain of their attributes retrieved) from the application to the hyperbase level, thus saving network communication time. Simulating existing HBMSs (such as Aalborg University's HyperBase) and developing existing hypermedia data models (such as the Dexter model) in Hyperform requires good knowledge of the existing systems and models. Hyperform developers must know about hypermedia objects, operations, and the exact functionality of the operations to be able to simulate the system (model). Since Hyperform classes are implemented in Scheme, a good knowledge of Scheme is required as well.

Hyperform has limited debugging support, since Elk has no integrated debugging facilities. When an error is encountered, Hyperform quits the operation, displays an error message, and continues to execute the next incoming message. Some debugging aids have been built into the basic classes of Hyperform. This makes it possible to trace errors to specific methods in classes and, thereafter, inspect variables. Ideally, errors should be corrected using a source-level debugger supporting breakpoints, stepping through single lines of the source code, examining and altering variables, etc.

The object-oriented approach to data modeling in Hyperform treats the metaclass, built-in classes, and data model classes as standard database objects. The object-oriented property supports the refinement of default system behavior using specialization and inheritance. The database property means that classes can be stored and accessed using the data-modeling facility. The Hyperform approach has three major advantages:

(1) *Extensibility.* Because the system itself is described with the object-oriented approach, it can be seen as a core of facilities that may be tailored to specific tools.
(2) *Accessibility.* Data held at the class level (metadata) are available like any other data in the system because classes are proper objects.
(3) *Uniformity.* The same tools are used to query, relate, and specialize data and metadata. Because there is no new mechanism to learn and no distinction made between data and metadata, the designer's only concern is the level of abstraction at which to work.

Normally, metaclasses do not facilitate kernel extensions, since they exist at the model level, above the structures that specify kernel services and policies (concurrency control, access control, etc.). By introducing kernel services as built-in classes, Hyperform allows developers to define policies at all hyperbase levels.

On the other hand, working with metaclasses can make Hyperform more difficult to understand, since the hypermedia system developer must cope with three levels of objects when designing hypermedia models: instances, classes, and the metaclass. Obviously, the Hyperform development environment is not intended to be used by a typical end-user or application programmer. Development of hypermedia models requires good knowledge of object-oriented techniques and expert knowledge of the hypermedia, hyperbase, and database fields. Even for a skilled hypermedia data management system developer, Hyperform requires different levels of expertise ranging from development of hypermedia data models through HBMS configurations to hypermedia system architectures. The three levels can be exemplified by (1) a Hyperform developer that creates a hypermedia data model using the built-in classes as is, (2) a Hyperform developer that specializes the built-in classes to develop a hypermedia data model, and (3) a Hyperform developer that creates a distributed hypermedia model based on multiple, distributed HBMSs.

As indicated, development of HBMSs in Hyperform is a quest for both flexibility and efficiency. Hypermedia system developers want Hyperform to support many (often exotic) features and at the same time provide fast and efficient storage, retrieval, and manipulation of Hyperform classes and objects. The first version of Hyperform provided a high degree of flexibility, but simply was not efficient enough to store large objects [Wiil 1993b]. The current version has been updated in several ways to solve this problem (such as using the Exodus Storage Manager to handle physical storage). Ongoing and future work (outlined in Section 7.1) will determine if these changes have given Hyperform a high degree of efficiency as well (first experiences so indicate).

The inclusion of the Exodus Storage Manager in the Hyperform was a step in the process of moving functionality from Scheme-based components to efficiently coded libraries. Exodus was primarily chosen for its capabilities to efficiently store and retrieve large binary objects (and because it is public domain software). Exodus provides some additional facilities that allowed functionality to be moved away from Scheme components, further enhancing the performance of Hyperform. Part of the functionality of Concurrency Control Object (short database transactions), Version Control Object (basic version storage handling), Query and Search Object (indexing), and System Object (caching and indexing) is now based on Exodus library functions.

The Exodus Storage Manager provides a set of general functions dealing with object storage management. This is especially useful in the Hyperform approach, since generality is a major design aspect of Hyperform. Hyperform makes extensive use of the Exodus library to provide a general set of hypermedia data management services. In addition, all Exodus functions are made available inside Hyperform to assist developers in the hypermedia-modeling process. This allows developers to combine the facilities of Exodus and other Hyperform components in new ways to address important hyperbase issues. For instance, the indexing facilities of Exodus can be combined with the built-in content search operations of QS Object to provide word-by-word content-based searches and structural searches.

6.2 Usability

The Hyperform development environment's intended application area is hypermedia. Within the area of hypermedia we can identify a number of potential applications, which indicate Hyperform's broad usability as a Hypermedia system development environment:

(1) *Link Engine for Intertool Linking* [*Kacmar and Leggett* 1991; *Meyrowitz* 1989; *Pearl* 1989; *Schnase* 1992; *Schnase et al.* 1993a]. Link engines store information on hypermedia associations and in some cases on the data involved in the associations. With an appropriate data model and HBMS configuration implemented, Hyperform can store any kind of data, including structure information. Thus, Hyperform can simulate existing link engines [Haan et al. 1992; Pearl 1989] and support distributed intertool linking. The runtime support for intertool linking would be located in the tool integrator in the Hyperform architecture.

(2) *Data Interchange between Existing Hypermedia Data Models* [*Leggett and Killough* 1991; *Leggett and Schnase* 1994]. More than one data model can be simulated in Hyperform simultaneously. Transformation (interchange) objects can be created, inheriting functionality from two data models and providing operations to convert from one format to the other (and vice versa).

(3) *Research Engine for Future Hypermedia Systems.* The rapid prototyping facilities ease development and experimentation with hypermedia data models, HBMS configurations, and hypermedia system architectures. New models and configurations can be derived from existing ones and shaped toward new application areas. To date, Hyperform has been used as an HBMS and hypermedia data model development tool. Ongoing and future work (outlined in Section 7.1) will use Hyperform to experiment with different hypermedia system configurations. As previously mentioned, the tool integrator is the central component of such experiments.

Hyperform also has research potential in other application domains:

(1) *Extensible OODB* [*Batory et al.* 1990; *Carey et al.* 1990; *Haas et al.* 1990; *Stonebraker and Rowe* 1986; *Wells et al.* 1992]. Hyperform can be classified as an extensible OODB, since we support common OODB features [Zdonik and Maier 1990] and introduce extensibility into basic database modules, such as concurrency control and version control. This view opens a wide range of possible application areas for Hyper-

form, such as office information systems, CSCW systems, software engineering systems, and programming systems. Since Hyperform can be extended and tailored on-the-fly, it can also be useful as a research engine in addressing future database issues (as discussed by Silberschatz et al. [1991]) in a fast prototyping manner.

(2) *Scheme OODB Extension.* Another way of classifying Hyperform is as an extension to Scheme supporting OODB facilities (persistent programming language facilities). This view opens new types of Scheme applications, since Hyperform supports persistent, sharable objects that can be composed of all manner of existing Scheme types and functions.

(3) *GNU Emacs Extension.* Existing GNU Emacs applications can be extended to make use of the Hyperform capabilities. The Hyperform send abstraction in Emacs enables access to the powerful Hyperform data-modeling facilities.

To increase the usability of Hyperform in hypermedia system development, the use of Hyperform could be combined with a hypermedia design method (such as the Enhanced Object-Relationship Model [Lange 1994]) to assist the hypermedia system developer in converting a problem specification into a detailed description of necessary hypermedia data model objects and operations. A hypermedia design method would give some useful guidelines in addressing steps (1) and (2) of a Hyperform development scenario (Section 4).

7. CONCLUSION

Development of HBMSs involves addressing certain critical issues. A data model must be determined, and decisions concerning the degree of concurrency control, notification control, access control, version control, and query and search support must be made. Once these policies are established the actual mechanisms must be implemented. The Hyperform development environment can be valuable in the policy-setting stage as well as the implementation stage. In the policy-setting stage, one might use the rapid prototyping features of Hyperform to quickly experiment with alternate designs. In the implementation stage, Hyperform provides the basic building blocks for effective implementation of the HBMS.

Efficiency and flexibility are often counteracting factors in system development. Therefore, HBMS designers often have to determine whether efficiency or flexibility should be the main goal of their work. It is important to notice that efficiency does not automatically rule out flexibility and vice versa. Reaching a design that has a high degree of both efficiency and flexibility should have a high priority in future HBMS development. Hyperform's design is an attempt to provide a high degree of both flexibility and efficiency. We have succeeded in addressing the flexibility aspect. Future application will determine if the current enhanced

Hyperform development environment succeeds in addressing the efficiency aspect as well.

The current trend in HBMS development is toward a high degree of openness, distribution, and heterogeneity in hypermedia system architectures. Hyperform is based on an extensible, open, and distributed architecture, and the HBMS server can accommodate several different hypermedia data models and HBMS configurations concurrently, thus addressing the heterogeneity aspect.

7.1 Ongoing and Future Work

The first version of Hyperform was fully implemented in the fall of 1992. Based on experiences with the first version, we have reimplemented most components of the system and incorporated the Exodus Storage Manager to promote efficiency and error handling.

We have an ongoing effort to create a hypermedia data definition language (DDL) and data manipulation language (DML) on top of Hyperform. The DDL allows Hyperform classes to be written in a more natural language (instead of Scheme), and the DML allows Hyperform data and structures to be queried and manipulated at a higher level of abstraction (without any knowledge of the underlying Scheme representation) [Krogsgaard et al. 1995].

Hyperform is currently providing a foundation for a hypermedia-based distributed collaborative computing environment (called HyperDisco) being developed at Aalborg University [Wiil 1995; Wiil and Leggett 1996]. HyperDisco is a distributed computing environment in which existing single-user tools can be integrated and extended to handle multiple collaborating users in a controlled manner. HyperDisco is based on an open hypermedia data model that supports intertool linking and multiple versions of shared artifacts. In connection with the HyperDisco project, we have the following plans for Hyperform:

(1) we will experiment with several system configurations in further addressing distribution and heterogeneity aspects of the Hyperform architecture;
(2) we will extend the Hyperform architecture to handle multiple heterogeneous hypermedia data repositories.

HyperDisco is an attempt to create a large-scale experimental system. Hopefully, the HyperDisco experiment can help determine the limits and bottlenecks of the current Hyperform development environment in terms of flexibility, efficiency, openness, distribution, heterogeneity, and scalability.

ACKNOWLEDGMENTS

Special thanks to Bob Allen, Paolo Paolini, and the three anonymous reviewers for their helpful comments. The authors gratefully acknowledge Randy Trigg for his thorough reading and comments on the dissertation from which much of this article is based [Wiil 1993b].

REFERENCES

AFRATI, F. AND KOUTRAS, C. D. 1990. A hypertext model supporting query mechanisms. In *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext.* Cambridge University Press, Cambridge, Mass., 52–66.

AKSCYN, R., MCCRACKEN, D., AND YODER, E. 1988. KMS: A distributed hypermedia system for managing knowledge in organizations. *Commun. ACM 31,* 7 (July), 820–835.

BARGHOUTI, N. S. AND KAISER, G. E. 1991. Concurrency control in advanced database applications. *ACM Comput. Surv. 23,* 3 (Sept.), 269–317.

BATORY, D. S., BARNETT, J. R., GARZA, J. F., SMITH, K. P., TSUKAUDA, K., TWICHELL, B. D., AND WISE, T. E. 1990. Genesis: A reconfigurable database management system. *IEEE Trans. Softw. Eng.* (Nov.), 1258–1272.

BERNSTEIN, P. A. AND GOODMAN, N. 1981. Concurrency control in distributed database systems. *ACM Comput. Surv. 13,* 2 (June), 185–221.

CAMPBELL, B. AND GOODMAN, J. 1988. HAM: A general-purpose hypertext abstract machine. *Commun. ACM 31,* 7 (July), 856–861.

CAREY, M., DEWITT, D., GRAEFE, G., HAIGHT, D., RICHARDSON, J., SCHUH, D., SHEKITA, E., AND VANDENBERG, S. 1990. The Exodus extensible DBMS project. In *Readings in Object-Oriented Databases.* Morgan Kaufmann, San Mateo, Calif., 474–499.

CAREY, M. J. ET AL. 1993. Using the Exodus Storage Manager. Dept. of Computer Science, Univ. of Wisconsin, Madison, Wisc. Available as ftp://ftp.cs.wisc.edu/exodus/sm/doc/sm3doc.ps.

CATLIN, T. J., BUSH, P., AND YANKELOVICH, N. 1989. InterNote: Extending a hypermedia framework to support annotative collaboration. In *Proceedings of the 2nd ACM Conference on Hypertext.* ACM, New York, 365–378.

CLINGER, W., AND REES, J., Eds. 1992. The revised^4 report on the algorithmic language Scheme. Tech. Rep. CIS-TR-91-25, Dept. of Computer and Information Science, Univ. of Oregon, Eugene, Oreg. Feb.

COHEN, E., SONI, D., GLUECKER, R., HASLING, W., SCHWANKE, R., AND WAGNER, M. 1988. Version management in Gypsy. In the *ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments.* ACM, New York, 201–215.

DAVIS, H., HALL, W., HEATH, I., HILL, G., AND WILKINS, R. 1992. Towards an integrated information environment with open hypermedia systems. In *Proceedings of the 4th ACM Conference on Hypertext.* ACM, New York, 181–190.

DELISLE, N. AND SCHWARTZ, M. 1986. Neptune: A hypertext system for CAD applications. In *Proceedings of the ACM International Conference on the Management of Data.* ACM, New York, 132–143.

FULLER, M., KENT, A., SACKS-DAVIS, R., THOM, J., WILKINSON, R., AND ZOBEL, J. 1991. Querying in a large hyperbase. In *Proceedings of the 2nd International Conference on Database and Expert Systems Applications.* Austrian Computer Society, 455–458.

FURUTA, R. AND STOTTS, P. D. 1990. The Trellis hypertext reference model. In *Proceedings of the Hypertext Standardization Workshop.* NIST, Washington, D.C., 83–93.

GARG, P. K. 1988. Abstraction mechanisms in hypertext. *Commun. ACM 31,* 7 (July), 862–870.

GRØNBÆK, K. AND TRIGG, R. H. 1992. Design issues for a Dexter-based hypermedia system. In *Proceedings of the 4th ACM Conference on Hypertext.* ACM, New York, 191–200.

GRØNBÆK, K., HEM, J. A., MADSEN, O. L., AND SLOTH, L. 1993. Designing Dexter-based cooperative hypermedia systems. In *Proceedings of the 5th ACM Conference on Hypertext.* ACM, New York, 25–38.

HAAKE, A. 1992. CoVer: A contextual version server for hypertext applications. In *Proceedings of the 4th ACM Conference on Hypertext.* ACM, New York, 43–52.

HAAN, B. J., KAHN, P., RILEY, V. A., COOMBS, J. H., AND MEYROWITZ, N. K. 1992. IRIS hypermedia services. *Commun. ACM 35,* 1 (Jan.), 36–51.

HAAS, L. M., CHANG, W., LOHMAN, G. M., MCPHERSON, J., WILMS, P. F., LAPIS, G., LINDSAY, B., PIRAHESH, H., CAREY, M., AND SHEKITA, E. 1990. Starburst Midflight: As the dust clears. *IEEE Trans. Knowl. Data Eng. 2,* 1 (Mar.), 143–160.

HALASZ, F. 1988. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Commun. ACM 31,* 7 (July), 836–852.

HALASZ, F. 1991. Hypertext '91 keynote talk. In the *3rd ACM Conference on Hypertext.* ACM, New York.

HALASZ, F. AND SCHWARTZ, M. 1994. The Dexter hypertext reference model. *Commun. ACM. 37,* 2 (Feb.), 30–39.

HICKS, D. L. 1993. A version control architecture for advanced hypermedia environments. Ph.D. dissertation, Texas A&M Univ., College Station, Tex.

KACMAR, C. J. AND LEGGETT, J. J. 1991. PROXHY: A process-oriented extensible hypertext architecture. *ACM Trans. Inf. Syst. 9,* 4 (Oct.), 399–419.

KAPLAN, S. M., CARROLL, A. M., LOVE, C., LALIBERTE, D. M., AND ANDREESSEN, M. 1992. Epoch—GNU Emacs for the X windowing system. Dept. of Computer Science, Univ. of Illinois, Urbana-Champaign, Ill. Available as http://src.doc.ic.ac.uk/gnu/epoch/epoch-4.2.tar.z.

KROGSGAARD, M., CHRISTIANSEN, M., RASMUSSEN, K. B., HAGEN, J. L., THOMASSEN, H. F., SKOV, K. Q., AND JENSEN, P. S. 1995. HAL—Hyperform Application Language. Dept. of Computer Science Int. Rep., Aalborg Univ., Aalborg, Denmark. In Danish.

LANGE, D. B. 1990. A formal model of hypertext. In *Proceedings of the Hypertext Standardization Workshop.* NIST, Washington, D.C., 145–166.

LANGE, D. B. 1994. An object-oriented design method for hypermedia information systems. In *Proceedings of the 27th Hawaii International Conference on System Sciences.* IEEE, New York, 366–375.

LAUMANN, O. 1993. Reference manual for the Elk extension language interpreter. Dept. of Computer Science, Indiana Univ., Bloomington, Ind. This document is part of the Elk system available from ftp://ftp.cs.indiana.edu/pub/scheme-repository/imp/elk-3.0.tar.gz.

LEGGETT, J. J. AND KILLOUGH, R. L. 1991. Issues in hypertext interchange. *Hypermedia 3,* 3, 159–186.

LEGGETT, J. J. AND SCHNASE, J. L. 1994. Viewing Dexter with open eyes. *Commun. ACM 37,* 2 (Feb.), 76–86.

LEGGETT, J. J., SCHNASE, J. L., SMITH, J. B., FOX, E. A., Eds. 1993. Final report of the NSF workshop on hyperbase systems. Tech. Rep. TAMU-HRL-93-002, Dept. of Computer Science, Texas A&M Univ., College Station, Tex. June.

MEYROWITZ, N. 1989. The missing link: Why we're all doing hypertext wrong. In *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information.* MIT Press, Cambridge, Mass., 107–114.

NOLL, J. AND SCACCHI, W. 1991. Integrating diverse information repositories: A distributed hypertext approach. *IEEE Comput. 24,* 12 (Dec.), 38–45.

ØSTERBYE, K. 1992. Structural and cognitive problems in providing version control for hypertext. In *Proceedings of the 4th ACM Conference on Hypertext.* ACM, New York, 33–42.

PEARL, A. 1989. Sun's link service: A protocol for open linking. In *Proceedings of the 2nd ACM Conference on Hypertext.* ACM, New York, 137–146.

RIZK, A. AND SAUTER, L. 1992. Multicard: An open hypermedia system. In *Proceedings of the 4th ACM Conference on Hypertext.* ACM, New York, 4–10.

ROHRBACH, R. AND SEIWALD, C. 1988. Galileo: A software maintenance environment. In *Proceedings of the International Workshop on Software Version and Configuration Control.* 444–456.

SCHNASE, J. L. 1992. HB2: A hyperbase management system for open, distributed hypermedia system architectures. Ph.D. dissertation, Texas A&M Univ., College Station, Tex.

SCHNASE, J. L., LEGGETT, J. J., HICKS, D. L., NÜRNBERG, P. J., AND SANCHEZ, J. A. 1993a. HB1: Design and implementation of a hyperbase management system. *Elec. Pub. Orig. Dissem. Des. 6,* 1 (Mar.), 125–150.

SCHNASE, J. L., LEGGETT, J. J., HICKS, D. L., AND SZABO, R. L. 1993b. Semantic data modeling of hypermedia associations. *ACM Trans. Inf. Syst. 11,* 1 (Jan.), 27–50.

SCHÜTT, H. A. AND HAAKE, J. M. 1993. Server support for cooperative hypermedia systems. In *Hypermedia—Proceedings der Internationalen Hypermedia '93 Konferenz.* Springer-Verlag, Berlin, 45–56.

SCHÜTT, H. A. AND STREITZ, N. 1990. HyperBase: A hypermedia engine based on a relational database management system. In *Hypertext: Concepts, Systems and Applications, Proceedings of the European Conference on Hypertext.* Cambridge University Press, Cambridge, Mass., 95–108.

SHACKELFORD, D. E., SMITH, J. B., AND SMITH, F. D. 1993. The architecture and implementation of a distributed hypermedia storage system. In *Proceedings of the 5th ACM Conference on Hypertext.* ACM, New York, 1–13.

SILBERSCHATZ, A., STONEBRAKER, M., AND ULLMAN, J. D., Eds. 1991. Database systems: Achievements and opportunities. *Commun. ACM 34,* 10 (Oct.), 110–120.

SMITH, J. B. AND SMITH, F. D. 1991. ABC: A hypermedia system for artifact-based collaboration. In *Proceedings of the 3rd ACM Conference on Hypertext.* ACM, New York, 179–192.

STALLMAN, R. M. 1984. EMACS: The extensible, customizable, self-documenting display editor. In *Interactive Programming Environments.* McGraw-Hill, New York, 300–325.

STONEBRAKER, M. AND ROWE, L. A. 1986. The design of Postgres. In *Proceedings of the ACM International Conference on Management of Data.* ACM, New York, 340–355.

TICHY, W. F. 1985. RCS—A system for version control. *Softw. Pract. Exper. 15,* 7 (July), 637–654.

TOMPA, F. W. 1989. A data model for flexible hypertext database systems. *ACM Trans. Inf. Syst. 7,* 1 (Jan.), 85–100.

WELLS, D. L., BLAKELEY, J. A., AND THOMPSON, C. W. 1992. Architecture of an open object-oriented database management system. *IEEE Comput. 25,* 10 (Oct.), 74–82.

WIIL, U. K. 1991. Using events as support for data sharing in collaborative work. In the *International Workshop on CSCW.* Institut für Informatik und Rechentechnik, Berlin, Germany, 162–176.

WIIL, U. K. 1992. Issues in the design of EHTS: A multiuser hypertext system for collaboration. In *Proceedings of the 25th Hawaii International Conference on System Sciences.* IEEE, New York, 629–639.

WIIL, U. K. 1993a. Experiences with HyperBase: A multiuser hypertext database. *SIGMOD Rec. 22,* 4 (Dec.), 19–25. HyperBase and EHTS are available from ftp://ftp.iesd.auc.dk/pub/packages/hypertext/HyperBase/readme.hyperbase.

WIIL, U. K. 1993b. Extensibility in open, distributed hypertext systems. Ph.D. dissertation, Tech. Rep. 93-2013, Dept. of Computer Science, Aalborg Univ., Aalborg, Denmark.

WIIL, U. K. 1995. HyperDisco: An object-oriented hypermedia framework for flexible software system integration. In *Proceedings of the 19th IEEE Annual International Computer Software and Applications Conference.* IEEE, New York, 298–305.

WIIL, U. K. AND LEGGETT, J. J. 1992. Hyperform: Using extensibility to develop dynamic, open and distributed hypertext systems. In *Proceedings of the 4th ACM Conference on Hypertext.* ACM, New York, 251–261.

WIIL, U. K. AND LEGGETT, J. J. 1993. Concurrency control in collaborative hypertext systems. In *Proceedings of the 5th ACM Conference on Hypertext.* ACM, New York, 14–24.

WIIL, U. K. AND LEGGETT, J. J. 1996. The HyperDisco Approach to Open Hypermedia Systems. In *Proceedings of the 7th ACM Conference on Hypertext.* ACM, New York, 140–148.

ZDONIK, S. B. AND MAIER, D., Eds. 1990. *Readings in Object-Oriented Database Systems.* Morgan Kaufmann, San Mateo, Calif.

ZOBEL, J., WILKINSON, R., THOM, J., MACKIE, E., SACKS-DAVIS, R., KENT, A., AND FULLER, M. 1991. An architecture for hyperbase systems. In *Proceedings of the 1st Australian Multi-Media Communications, Applications and Technology Workshop.* 152–161.

Received March 1994; revised November 1994 and August 1995; accepted November 1995