The HyperDisco Approach to Open Hypermedia Systems

Uffe Kock Wiil

Department of Computer Science
Aalborg University
Fredrik Bajers Vej 7E
9220 Aalborg Øst, Denmark
Email: kock@iesd.auc.dk

John J. Leggett

Hypermedia Research Laboratory
Department of Computer Science
Texas A&M University
College Station, Texas 77843-3112, USA
Email: leggett@bush.cs.tamu.edu

ABSTRACT

Computing support for large engineering enterprises provides an example of the need for hypermedia-based collaborative computing systems composed of a large number of distributed heterogeneous tools. These computing environments place complex requirements on the underlying hypermedia platform. To support integration of independently written tools for these environments, hypermedia platforms must address several important issues such as scalability, openness, distribution, heterogeneity, interoperability, extensibility and computation.

This paper describes the HyperDisco approach to open hypermedia systems. HyperDisco provides an extensible object-oriented hypermedia platform supporting inter-tool linking, computation, concurrency control, notification control, version control, access control, query and search, and various other features. The present work has two main objectives: 1) to provide a platform to integrate existing and future distributed heterogeneous tools and data formats and 2) to provide a platform to extend integrated tools to handle multiple collaborating users and multiple versions of shared artifacts. The paper presents important dimensions of hypermedia platforms that helped to formulate the goals for HyperDisco, the HyperDisco prototype, and two integration examples to illustrate the distinctive features of the HyperDisco approach.

KEYWORDS: Open hypermedia systems, integration, hypermedia platforms, collaborative work, system architectures, data models, inter-tool linking, link services, hyperbase management systems, scalability, openness, distribution, heterogeneity, interoperability, extensibility, computation

1 INTRODUCTION

Open systems and integration of information are two of the critical issues in information systems today. Current computing environments consist of a diversity of programs, systems and applications (tools) each supporting different tasks in our daily work, such as software development, design, authoring, planning and communication. Typically, few means exist for collaboration or integration among the tools. Each tool has its own standard for information storage and retrieval. Consequently, information is stored in different formats and located in distributed heterogeneous information repositories. The lack of integration among tools and the information maintained by the tools has been identified as a major problem inherent in the current generation of computing environments [2, 11, 12].

Hypermedia linking provides a powerful solution to integration in existing and future computing environments, both with regard to the integration of different tools and the integration of different data formats and data exchange. We have seen a rapidly growing interest in the design, development and deployment of open hypermedia platforms such as Sun's Link Service [13], Microcosm [2, 3], Proxhy [8], Multicard [14], DHM [4, 5], Hyperform [19], SP3 [9], and Chimera [1]. An open hypermedia platform supports inter-tool linking as a means of integrating distributed heterogeneous tools and data formats. System developers are supplied with an open system architecture including a link service and a storage subsystem typically referred to as a hyperbase management system (HBMS). The link service provides a communication protocol allowing third-party distributed heterogeneous tools to participate in hypermedia services (anchoring and linking capabilities). The HBMS provides efficient storage and retrieval of anchoring, linking and other hypermedia information. In addition, hyperbases manage a rich and complex set of data types, dynamically manage the structure of the data (metadata), support multiple collaborating users and provide powerful navigation facilities to support both hypermedia and third-party tools [10, 15, 18, 20]. Ideally, an open hypermedia platform should support the integration of

all manner of existing and future tools, including tools that: 1) have been developed independently of the hypermedia platform (third-party tools); 2) support single users, multiple users and collaboration among users; 3) support different data formats; 4) support a variety of application areas; 5) run on different hardware platforms; and 6) are distributed across different computing networks.

Hypermedia-based collaborative computing environments composed of a large number of distributed heterogeneous tools places complex requirements on the underlying hypermedia platform [5, 11]. This paper describes the HyperDisco approach to open hypermedia systems. HyperDisco provides an extensible object-oriented hypermedia integration model supporting inter-tool linking, computation, concurrency control, notification control, version control, access control, search and query, and various other features. The present work has two main objectives: 1) to provide a platform to integrate existing and future distributed heterogeneous tools and data formats and 2) to provide a platform to extend integrated tools to handle multiple collaborating users and multiple versions of shared artifacts.

In Section 2 we enumerate and briefly discuss important dimensions of hypermedia platforms that helped to formulate the goals for HyperDisco. Section 3 presents the HyperDisco prototype and Section 4 gives two integration examples illustrating the distinctive features of HyperDisco. Section 5 compares HyperDisco to related open hypermedia systems research. Finally, Sections 6 and 7 give our conclusions and a brief plan for future work.

2 HYPERMEDIA PLATFORM DIMENSIONS

Hypermedia platforms vary along several dimensions [9, 10, 15, 18]. To be able to compare and characterize current and future hypermedia platforms, we describe seven major dimensions. Each dimension relates to both the system architecture and HBMS of the hypermedia platform.

1. Scalability. Scale refers to the number of simultaneous users and tools effectively handled by the system architecture. The number may vary from one to tens of thousands. Scale also refers to the size of the HBMS storage both in terms of the number of bytes and number of objects managed. Scale ranges from small sizes of a few megabytes, through medium sizes of a few hundred gigabytes, to large sizes of hundreds of terabytes (and from a few hundred, through tens of thousands, to millions of objects) for digital libraries.

2. Openness. A closed hypermedia platform is composed of a fixed set of encapsulated tools. These tools are normally tightly integrated with the HBMS. An open hypermedia platform provides a protocol that allows any tool to participate in hypermedia services. These tools are loosely integrated with the HBMS. Openness in system architectures ranges from no hypermedia data model restriction on the tools, to the situation in which the data model is embedded in the underlying HBMS and imposed by the protocol on the tools. Openness in HBMS ranges from a multi-session extended client-server approach, to the situation in which the HBMS facilities are statically linked with the tools.

3. Distribution. In terms of the system architecture, this dimension describes where the tool and hypermedia platform processes execute. Distribution ranges from all processes executing on the same machine (centralized), to the situation in which all processes can execute on different machines in a network (massively distributed). In terms of the HBMS, this dimension describes whether the data managed by the HBMS resides on one machine's store (centralized) or on multiple machines' stores (distributed).

4. Heterogeneity. In a homogeneous hypermedia platform, participating tools share the same hypermedia data model. In a heterogeneous hypermedia platform, participating tools may have different data models. In terms of the HBMS, this dimension describes whether the same (homogeneous) or different (heterogeneous) hypermedia data models are supported by the HBMS. A multi-HBMS may be distributed and heterogeneous.

5. Interoperability. This dimension describes whether the system architecture is capable of using one or several protocols. Since no standard protocol exists, interoperability ranges from no protocol (closed), to the support of several protocols (open). Interoperability in HBMS ranges from no exchange formats supported (closed), to several exchange formats supported (open). Ideally, commonly agreed upon standards for interoperability and data exchange should be supported.

6. Extensibility. This dimension describes whether new tools, data formats and protocols can be added to the system architecture. Dynamic extensibility allows tools, data formats and protocols to be added at run time. Extensibility in HBMS relates to whether the data model can be extended by adding new data abstractions and operations. Dynamic extensibility allows the data model modification to occur at run time.

7. Computation. This dimension describes whether the platform supports computation of data model elements such as nodes, links, anchors and virtual structures. Computational capacity is provided by hypermedia data model abstractions and can occur at different architectural levels in the hypermedia system, including the HBMS.

While these dimensions can be used to characterize and compare existing hypermedia platforms, they can also serve as design goals for new hypermedia platforms. In the design of HyperDisco, we attempted to place its characteristics on the high end of all seven dimensions. For example, HyperDisco should be open and extensible at all levels, massively distributable and highly scalable. The HyperDisco name is derived from its intended use as a hypermedia platform for distributed collaborative computing environments.

3 HYPERDISCO

The overall goal of the HyperDisco project is to provide a hypermedia platform for flexible integration and extension of tools. The basic idea in this approach is to allow different tools to be integrated in HyperDisco at different tool-dependent levels. Instead of providing a single model of integration that all tools must adhere to, we allow each tool or tool group to have its own specialized model of integration and its own specialized protocol for accessing hypermedia services. An integrated tool can make use of as many (or as few) of the provided hypermedia services as desirable.

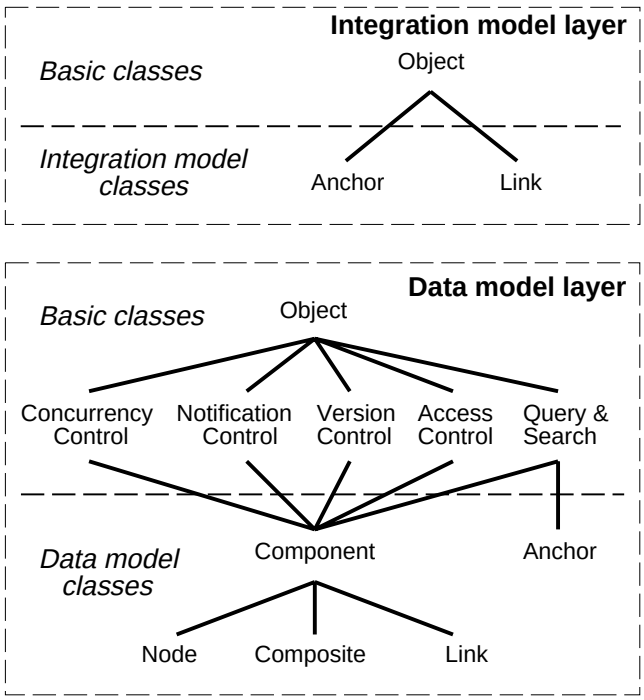

Figure 1: The HyperDisco layered hypermedia platform. Integration support for participating tools is provided in the integration model layer and storage support is provided in the data model layer.

HyperDisco provides two distinct layers of hypermedia functionality as depicted in Figure 1. The integration model and data model layers provide a set of general-purpose built-in services (a library of reusable object-oriented software routines). In the integration model layer, the built-in classes provide basic hypermedia linking services (anchors and links) including a flexible communication protocol. In the data model layer, the built-in classes provide basic hypermedia storage services for hypermedia objects (nodes, composites, links and anchors). The integration model layer encapsulates tool-dependent hypermedia linking functionality and provides an interface to the general hypermedia functionality of the data model layer. Individually tailored tool interfaces and communication protocols are created by specializing the behavior of the basic hypermedia linking services. The integration model layer provides functionality only[1] and relies (via message passing) on the data model layer functionality to provide many essential operations such as creation, storage, retrieval, updates, queries and deletion of hypermedia objects.

Both layers can be extended and tailored using (multiple) inheritance to provide specialized integration models (and storage support) for individual tools or tool groups. Thus, HyperDisco allows development (and concurrent operation) of different integration models that to a large extent share the same basic data model functionality. The remainder of this section describes the two layers and the HyperDisco prototype implementing these two layers.

3.1 Data Model

HyperDisco provides an extensible object-oriented data model with a set of general hypermedia object types: Anchor, Node, Link and Composite (Figure 1). The Component class inherits facilities from five basic classes: Concurrency Control, Notification Control, Version Control, Access Control and Query&Search. The Anchor class is a subclass of the Query&Search class. New (sub)types can be added to match specific storage requirements of participating tools. For example, an anchor subtype providing access control can be created by making a new class that inherits from the Anchor and Access Control classes. In this way, participating tools can extend and tailor the data model by adding new methods and specializing existing methods.

All instances in HyperDisco have a unique object identifier (OID). Anchors (inspired by Dexter [7]) simply contain a value that specifies some location, region, item, or substructure within a component. If the anchor value is empty, the anchor is a whole-component anchor [4]. The smallest sharable unit in the model is an attribute. Components are collections of attributes. Some attributes are always present in components and additional attributes can be attached dynamically. Default component attributes are a list of anchor OIDs, an access con-

[1] The integration model layer classes have no instances. Hypermedia objects belong to the data model layer.

trol list, version information and additional information such as owner, time created and last updated. Components and anchors can have attached scripts that extend the behavior of instances [14].

The data model supports a node subtype for each media type (text, graphics, video, voice, etc.) supported by HyperDisco. For instance, the text node subtype provides operations to deal with text searches (regular expressions over attribute contents). Adding a new media type to HyperDisco involves the creation of a node subtype, specification of at least one editor / browser tool that can display the media type (in a default node attribute named application), and addition of query and version compression (delta) operations for the media type. To allow integration of monolithic third-party tools, nodes can be used as data wrappers [4] or proxhy objects [8]. Instead of storing the contents, a data wrapper node stores a reference to the actual location of the contents. This allows links to be anchored to files in the Unix file system. Data wrapper nodes launch the appropriate display tools to display the endpoints of traversed links.

The Link class supports a very general multi-headed (n-n) link. Links have a direction, but can be traversed in both directions. Links maintain two lists of endpoints (to and from). A link endpoint can be one of three types: static, dynamic (computed) or dangling. A static endpoint contains an anchor / component OID pair, a dynamic endpoint contains a script evaluating to an anchor / component OID pair, and a dangling endpoint is simply empty. Links are either private, group or public, depending on the value of the access control attribute. Links contain an attribute (trigger:) that is evaluated before the link is traversed. Thus, link traversal can trigger arbitrarily complex internal computations and execution of external processes. We support a variety of link subtypes such as static 1-1 links and dynamic 1-n links.

The Composite class supports a general aggregation mechanism. Composites are unstructured collections (sets) of other components referenced by their OID. Composites are either static or dynamic (computed). A static composite contains a list of component OIDs, while a dynamic composite contains a script (query) evaluating to a list of component OIDs. Different composite subtypes are supported such as virtual composites capable of storing transient results of (structural) queries [6] and structured collections.

The Component class (and thus its three subclasses: Node, Link and Composite) provides support for collaboration, query and search, and version and access control:

Collaboration Support. HyperDisco provides short database transactions combined with user-controlled locking [20]. User-controlled locks provide support for long duration updates to objects in the database and are shared, fine-grained (attribute-level) and persistent. A collaborative update session in HyperDisco is initiated with an explicit lock request. Each operation in the session (such as frequent saves) are performed within short database transactions in the underlying storage manager. Locked objects can be read by other users and granted locks are stored persistently. In this way, HyperDisco is able to recover from both client and server crashes within long collaborative sessions. Events from HyperDisco enable participating tools to monitor changes in shared hypertexts. The concurrency control and notification control mechanisms operate at the attribute level, allowing attributes in a component to be locked independently of each other and allowing monitoring of operations on individual attributes.

Query and Search Support. HyperDisco supports both content and structure based queries as a supplement to navigational information retrieval (browsing and link traversal). Content search operations are dependent on the media type and, thus, supported in node subtypes (e.g., regular expressions over attribute contents in text nodes). Structure search operations are built on top of content search operations (all structural information is stored in attributes). A small generic (type independent) set of structure search operations are supported in the Component class. We support operations to determine types of instances and types of links between specific types of instances. Basic structure search operations (and content search operations) can easily be combined and extended to support more powerful structural queries. Boolean operators such as NOT, AND and OR and control structures such as IF, COND and CASE are used to glue basic search operations together.

Version and Access Control. Components in HyperDisco may be versioned. Versioning of (node) contents are directly supported and the necessary facilities for versioning of structure are present, but this feature is not directly supported. Participating tools explicitly request new versions (old versions are frozen). In this way, individual tools make use of the version facilities. Non-versioning tools will make all updates to a single version. Components maintain an access control list giving individuals, groups or all legal users read, write, annotate and delete privileges. The version control and access control mechanisms work at the component level.

3.2 Integration Model

The basic built-in integration model supports two types of hypermedia objects, Anchor and Link (Figure 1). The Anchor class provides operations to create, update and

delete endpoints in documents. The Link class provides operations to create, update, delete and follow links.

Participating tools can extend and tailor the integration model by adding new (sub)types and new methods, and specializing existing methods in the Anchor and Link class. This means that tools can agree on their own models for increased (or decreased) integration and participation in hypermedia services. The integration model layer also provides a flexible communication protocol supporting an extensible set of hypermedia services rather than a fixed set of services. Communication between participating tools and HyperDisco is based on a generic object-oriented message passing mechanism: (send object message . arguments).

To allow integration of tools that have been designed to work closely with HyperDisco, as well as integration of third-party tools, tools can participate in the hypermedia services at different levels. To illustrate this, we describe three levels of integration: full, partial and none.

Full Integration. Tools use HyperDisco to store all documents in different node types. Documents managed by the tools can serve as endpoints of links. Anchors allow pieces of data within documents to serve as endpoints for links. When navigating a link to a document, HyperDisco launches the appropriate tool displaying the document. If the anchor connects the link to a particular piece of information in the document, this piece of information is highlighted in the tool's interface.

Partial Integration. Tools use HyperDisco to store anchors, links and nodes that connect (pieces of) documents which are stored elsewhere in the computing environment (e.g., in the file system). Since anchors and links are stored separately and superimposed on the documents by the tools, linking is possible into documents stored on read-only media such as CD-ROM.

No Integration. Tools do not use HyperDisco for storage. Documents managed by non-integrated tools can only have incoming links; it is not possible to follow links from these documents. Anchors, if provided, can only refer to a document as a whole, not to a particular piece of data within the document. When navigating a reference to a document managed by a non-integrated tool, HyperDisco launches the tool which then displays the document. No hypermedia services are available when working with this tool.

Depending on the level of integration, HyperDisco places certain requirements on tools in order to use hypermedia services. Requirements on tools range from no requirements (no integration), to the ability to communicate with HyperDisco (e.g., using sockets) and exchange information (partial integration), to the situation where the tool relies 100% on the services of HyperDisco (full integration). A fully integrated tool must be able to communicate storage requests to HyperDisco and respond to hypermedia events. Fully integrated tools can bypass the integration model layer and communicate directly with the data model layer when making storage requests, like in the SP3 system [9]. In the partial integration case, the exact information exchanged depends on the chosen model of integration. The model must specify a way to uniquely identify the appropriate link in a follow-link request. Depending on the model, the tool could, for instance, communicate an OID of the selected anchor or an anchor value / file name pair to allow HyperDisco to determine which link to follow. In addition to these requirements, tools supporting hypermedia linking must in some way make these facilities available to the user through a user-friendly interface. The user should be able to locate anchors and follow links to related documents and create, share and delete relations in a natural way.

3.3 Prototype

The HyperDisco architecture is composed of distributed HBMSs, tool integrators and participating hypermedia and third-party tools as depicted in Figure 2. Tool integrators implement the integration model layer and HBMSs implement the data model layer. The architecture can be scaled from a single tool integrator / HBMS pair to multiple, distributed tool integrators and HBMSs.

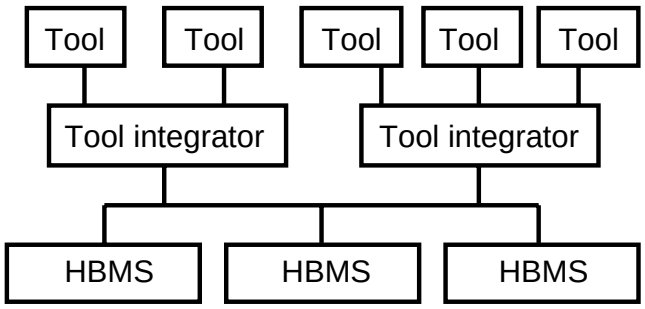

Figure 2: The HyperDisco architecture. Tool integrators provide the integration model layer and hyperbase management systems (HBMSs) provide the data model layer of HyperDisco.

The tool integrator is the central component of the architecture; it supports multiple integration models that allow distributed heterogeneous tools to participate in hypermedia services. The tool integrator also controls access and operation of multiple HBMSs and implements all the necessary runtime support for participating tools (such as distribution of events and storage, manipulation and retrieval of linking and anchoring information from the HBMSs). Tool integrators also allow participating tools (connected to the same tool integra-

tor) to share hypermedia structures and to communicate using an event-based mechanism.

Each instance of the HBMS can be thought of as a workspace. Since each HBMS instance maintains a list of legal users, a workspace can be public, private or belong to a group. Workspaces are partitioned into substructures (composites) which can be private, public or belong to a group. Thus, a workspace can be shared among collaborating users. Users can have multiple open workspaces and create links between documents residing in different workspaces. To enter a different workspace (e.g., by traversing a link), users (or tools) request that the tool integrator make a connection (if the HBMS server in question is running) or start a new HBMS instance that can provide the workspace. Currently, HyperDisco does not support any replication among the HBMSs.

The tool integrator and the HBMS components are based on an internal computational engine that provides an object-oriented extension language which allows the built-in services to be extended and tailored dynamically at runtime. New (versions of) classes are created by composing and sending a message to the component containing a description of the class. Thus, the integration model and data model can be customized at runtime, allowing tools to be integrated on the fly in a rapid prototyping manner.

HyperDisco operates in Unix environments and is presently hosted on Sun Sparcstations. Currently, we have integrated a small set of tools and data formats to assess the integration and collaboration capabilities of HyperDisco (see Section 4). HyperDisco was implemented by using the Hyperform hypermedia system development environment [19].

4 INTEGRATION AND EXTENSION EXAMPLES

Integration of tools in HyperDisco will generally require the following steps: analyzing the tool to determine its degree of openness, determining an appropriate level and model of integration for the tool, and specializing the built-in functionality of HyperDisco to support the chosen level and model of integration. The latter task is divided into two subtasks: specializing the integration model and specializing the data model of HyperDisco to provide the necessary services.

In this section, we describe two tests on the integration and extension of a small selection of tools supporting authoring: a textual editor (GNU Emacs [16]), a document processing system (LaTeX), a dvi display tool (Xdvi), a conversion script (dvips), a postscript display tool (PageView) and a printer script. In the first test, we used an Anchor Link integration model to integrate the selected set of tools. In the second test, we used a Data Wrapper Node integration model to integrate and extend the same set of tools to support multiple collaborating users and multiple versions of documents.

In both tests, we made Emacs the central tool in the integrated authoring environment. Within Emacs, users can traverse links from the textual representation to both display tools (Xdvi and PageView). Traversing a link from Emacs to either display tool causes the textual representation to be translated into an updated dvi or postscript file (using the available scripts if necessary) and activation of the display tool with the appropriate file. Users can also activate a print link in Emacs that generates an updated postscript file (if necessary) and sends it to the printer. In addition to the three default computational links (xdvi, pageview and print), which are available from all LaTeX buffers in Emacs, users can create links to all types of files from Emacs (e.g., to postscript figures created by the Idraw drawing tool [17]).

4.1 Test 1: Integration of Tools

In the first test, Emacs is partially integrated with HyperDisco using an integration model composed by anchor and link abstractions as depicted in Figure 3. Anchors are made responsible for maintaining information on where the document is stored, which application can display the document, and which particular piece of data within the document (if any) is used as endpoint for the link. Links in the model are simply used to connect anchors. This integration model required Emacs (as a partially integrated tool) to be capable of managing an anchor table data structure and communicating with HyperDisco via sockets. The contents of the anchor table is retrieved from HyperDisco when a file is loaded. The model places no requirements on Idraw as a non-integrated tool.

The following example illustrates the communication (message passing) necessary to follow a link from Emacs to Idraw using the Anchor Link integration model:

Follow a link from Emacs to Idraw. By selecting the anchor value "hypermedia link", the user indicates that she wishes to follow the link associated with the anchor. Emacs looks up the anchor value in the internal anchor table and finds the anchor OID (#431). Emacs sends a **follow-link** message to the link class in the tool integrator with the anchor OID as argument: *(send integration-model-link-class follow-link 431)*. The **follow-link** method sends a message to the HBMS requesting the anchor associated with anchor #431: *(send data-model-link-class get-anchor 431)*. The **get-anchor** method locates the link associated with anchor #431 (by querying the database), this is link #1608. The **get-anchor** method reads the associated anchor OID (#142) and then reads and returns the entire anchor object (#142)

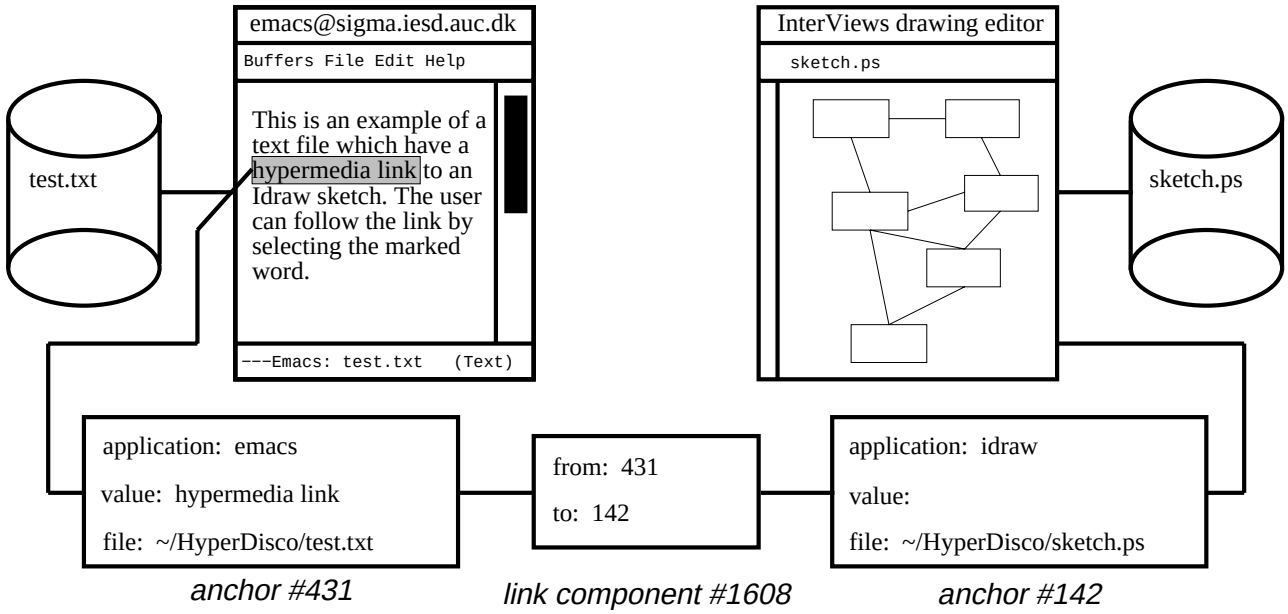

Figure 3: Principles and abstractions in the Anchor Link integration model.

to the calling method (**follow-link**). Based on the attribute values of the anchor object (#142), the **follow-link** method starts the Idraw application which displays the relevant file; this completes the **follow-link** operation.

Implementation of the Anchor Link integration model in HyperDisco involved specialization of the built-in Anchor and Link classes in the integration model layer and the built-in Anchor and Link classes in the data model layer to provide the necessary services (attributes and methods). It was, for instance, necessary to extend the data model Link class with a **get-anchor** method and the data model Anchor class with additional attributes (application and file).

4.2 Test 2: Integration and Extension of Tools

In the second test, we extended the authoring environment to handle multiple collaborating users. We had to incorporate the use of additional features such as access control, notification control and concurrency control into participating tools. Therefore, the second test uses a more complicated integration model with node, link and anchor abstractions (Figure 4) to partially integrate Emacs. Anchors simply hold information on what particular piece of data in the document (if any) is used as a link endpoint. Nodes are now made responsible for maintaining information on where the document is stored, which application can display the document, and which anchors are associated with the document. Links are used to connect node / anchor pairs.

The move from a single user to a collaborative authoring environment required an increased integration of Emacs into HyperDisco. HyperDisco stores nodes, links and anchors and maintains locking, notification, and access control information. Emacs still uses the Unix file system to store files. To have a point of reference within HyperDisco for multiuser operations (e.g., locking a file and subscribing to updates to a file) and a place to store additional attributes, we have a data wrapper node in HyperDisco for each file available in the authoring environment. In this scenario, Emacs is engaged in a great deal of communication with HyperDisco, since most of the extensions reside in HyperDisco. For instance, before accessing a node or traversing a link, the authoring environment must check for locks, events and appropriate access rights. Based on the reply, Emacs can either start the operation or communicate an exception to the user.

Different integration models place different requirements on tools in order to be partially integrated in HyperDisco. Models that place more requirements on the tools rely less on queries in the HBMS. For example, the fact that Emacs in the first test is able to communicate the anchor OID to the integration model link class saves two complex queries in the HBMS (in a **follow-link** operation), compared to a situation where the tool is only capable of communicating the anchor value, the file and the application. Therefore, in the second test, we had Emacs maintain an anchor table and the node OID (both are retrieved from HyperDisco when a file is loaded).

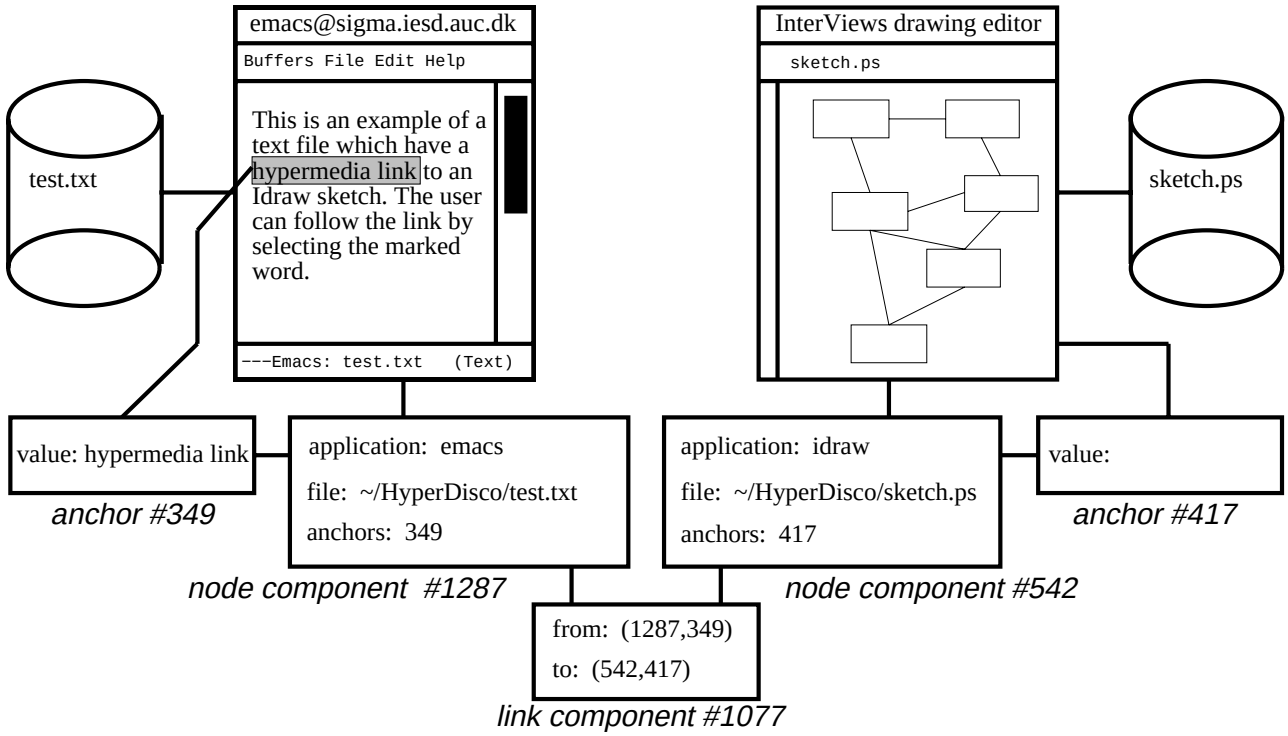

Figure 4: Principles and abstractions in the Data Wrapper Node integration model.

The following example illustrates the communication (message passing) necessary to follow a link from Emacs to Idraw using the Data Wrapper Node integration model:

Follow a link from Emacs to Idraw. When an anchor has been activated (in this case "hypermedia link"), Emacs looks up the anchor value in the internal anchor table and finds the anchor OID (#349). Emacs sends a **follow-link** message to the link class in the tool integrator with the anchor and node OIDs as arguments: *(send integration-model-link-class follow-link 349 1287).* The **follow-link** method sends a message to the HBMS requesting the node associated with the selected endpoint: *(send data-model-link-class get-node 349 1287).* The **get-node** method locates the link associated with the selected endpoint (by querying the database), this is link #1077. The **get-node** method reads the associated node OID (#542) and then reads and returns the entire node object (#542) to the calling method (**follow-link**). Based on the attribute values of the node object (#542), the **follow-link** method starts the Idraw application which displays the relevant file; this completes the **follow-link** operation.

Implementation of the Data Wrapper Node integration model in HyperDisco involved adding a Data Wrapper Node class (integration model layer), specialization of the built-in Anchor and Link classes (integration model layer) and specialization of the built-in Node, Link, Component and Anchor classes (data model layer).

We went a step further and extended the authoring environment to handle multiple versions of shared documents. While this can be done in many different ways, we decided to version only the LaTeX nodes, since the other document formats can be derived (computed) from the LaTeX format. When a new version of a LaTeX node is generated by an explicit command in Emacs, links from the old version are reused in the new version. This raises an interesting issue. Should the old versions of the document reside as files in the file system (using some version numbering mechanism) or should they reside in the HBMS? We decided to store the latest version of a document in the file system and old versions in nodes in the HBMS. This solution makes the versioning extension transparent from a file system perspective and ensures that only the latest version of the document can be corrupted by non-participating tools. Nodes are grouped into version groups and old versions can be retrieved by the query mechanism based on different attributes such as last updated, time created or owner.

5 COMPARISON TO RELATED WORK

Existing open hypermedia platforms can be divided into two categories, open hyperbase and link server [21]. The

open hyperbase category (DHM [4, 5] and SP3 [9]) includes a HBMS supporting a full hypermedia data model capable of providing storage to hypermedia tools as well as providing storage of connectivity information to third-party tools. The link server category (Microcosm [2, 3], Proxhy [8], Multicard [14], Sun's Link Service [13] and Chimera [1]) makes the assumption that participating tools use other information repositories to store their documents and only use the hypermedia platform to store and retrieve connectivity information. This latter type of platform includes a link database instead of a HBMS.

HyperDisco belongs to the open hyperbase category. HyperDisco provides a full hypermedia data model and allows tools to participate in the hypermedia services at various levels depending on the tool. Compared to the other open hyperbase approaches (SP3 and DHM), HyperDisco has most features in common with DHM. While DHM is based on the Dexter model, the development of the HyperDisco data model layer has been inspired by the Dexter model abstractions. We have tuned the concepts of components, nodes (atoms in Dexter), composites, links and anchors toward an open hypermedia systems philosophy. Both systems have added data model facilities for multiple collaborating users and query and search and use a layered system architecture consisting of participating tools, a runtime layer and a HBMS layer. In contrast to this, the conceptual model of hypermedia underlying SP3 is a process model allowing for distribution of hypermedia services and components across wide-area networks and heterogeneous platforms. SP3 is based on a layered system architecture consisting of participating applications, link services managers, and instances of the HB3 HBMS. HB3 is composed of an association set manager, an object version manager and a storage manager. HB3 maintains a rich hypermedia data model with support for multiple collaborating users. Like SP3, the HyperDisco HBMS supports version control, allows arbitrarily complex computations when traversing links[2], and allows the information space to be divided into several workspaces, each supported by its own HBMS instance.

Most open hypermedia platforms provide a single integration model that all tools must adhere to. For example, Microcosm provides a simple integration model providing keyword (local and generic) and positional (specific) links [2]. Microcosm places a strong emphasis on computation and separation of connectivity information from documents (links are stored in linkbases). Users interact with "viewers". Viewers translate user actions into messages that pass through an extensible chain of filters. Messages requesting that a link be followed are handled by linkbase filters. Different types of third-party viewers (ranging from fully aware to partially aware to totally unaware viewers) can be integrated by different means such as source code adaptation, creation of hypertext macros in the viewers, and the use of tool wrappers [3].

Like Microcosm, HyperDisco supports different means for integrating heterogeneous third-party tools at different levels of participation. In the examples described in Section 4, Emacs was partially integrated with HyperDisco by a combination of source code adaptation and macro extension. HyperDisco allows the hypermedia services to be dynamically extended and tailored thus supporting a rapid prototyping approach to tool integration. HyperDisco differs from other open hypermedia platform approaches in at least two ways: 1) by allowing both the integration model and data model to be arbitrarily customized towards specific tools, and 2) by supporting multiple integration models and data models. These features all add to the tool integration flexibility of HyperDisco.

6 CONCLUSION

We have presented the HyperDisco open hypermedia system approach to integration of information in current and future computing environments. HyperDisco allows distributed heterogeneous third-party tools to participate in hypermedia services at various tool-dependent levels. Participating tools can agree on their own protocols for increased (or decreased) integration and participation in the hypermedia services. HyperDisco provides a platform to extend participating tools to support multiple collaborating users and multiple versions of shared documents.

Tests have provided a proof of principle for HyperDisco's tool integration and tool extension capabilities. To allow flexible integration of (third-party) tools, HyperDisco provides an extensible integration model framework allowing each participating tool (or collection of tools) to implement its own customized integration model. In addition, Hyperdisco is not limited to one integration model. HyperDisco supports rapid prototyping of hypermedia services and allows multiple integration models to co-exist concurrently, which can be useful in future research on hypermedia integration.

7 FUTURE PLANS

We will continue to use HyperDisco in various test settings and integrate more tools and data formats, thereby gaining valuable experience with integration in open, extensible hypermedia-based computing environments. Future research directions include experimentation with workspaces, tool integrators, participating tools and in-

[2] While SP3 achieves this through its anchor and link processes, HyperDisco (like Multicard) includes an interpreter-based scripting language which allows computational behavior to be attached to its components and anchors.

formation repositories distributed across network-based heterogeneous hardware platforms. Tests will be used to identify issues to be addressed in future research and to further assess the capabilities of the HyperDisco prototype in terms of scalability, openness, distribution, heterogeneity, interoperability, extensibility and computation. Current experiences indicate that HyperDisco is on the high end of the latter six dimensions. To assess HyperDisco's scalability, we plan to perform large-scale experiments that use HyperDisco as a basis for digital libraries.

ACKNOWLEDGMENTS

This research was supported in part by the Danish Natural Science Research Council through Programs 9400911 and 9500996 and the Texas Advanced Research Program under Grant No. 999903-230.

REFERENCES

1. Anderson, K.M., Taylor, R.N., and Whitehead, E.J. Chimera: Hypertext for heterogeneous software environments. In *Proceedings of ECHT'94*, ACM Press, 1994, pp. 94–107.

2. Davis, H., Hall, W., Heath, I., Hill, G., and Wilkins, R. Towards an integrated information environment with open hypermedia systems. In *Proceedings of ECHT'92*, ACM Press, 1992, pp. 181–190.

3. Davis, H., Knight, S., and Hall, W. Light hypermedia link services: A study of third party application integration. In *Proceedings of ECHT'94*, ACM Press, 1994, pp. 41–50.

4. Grønbæk, K., and Trigg, R.H. Design issues for a Dexter-Based hypermedia system. *Commun. ACM*, 37, 2 (Feb. 1994), 40–49.

5. Grønbæk, K., Hem, J.A., Madsen, O.L., and Sloth, L. Cooperative hypermedia systems: A Dexter-based architecture. *Commun. ACM*, 37, 2 (Feb. 1994), 64–74.

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

7. Halasz, F., and Schwartz, M. The Dexter hypertext reference model. *Commun. ACM*, 37, 2 (Feb. 1994), 30–39.

8. Kacmar, C.J., and Leggett, J.J. Proxhy: A process-oriented extensible hypertext architecture. *ACM Trans. Inf. Sys.*, 9, 4 (Oct. 1991), 399–419.

9. Leggett, J.J., and Schnase, J.L. Viewing Dexter with open eyes. *Commun. ACM*, 37, 2 (Feb. 1994), 76–86.

10. Leggett, J.J., Schnase, J.L., Smith, J.B., and Fox, E.A., Eds. Final report of the NSF workshop on hyperbase systems, Washington, D.C., October 15-16, 1992. Department of Computer Science Technical Report TAMU-HRL-93-002, Texas A&M University, College Station, Texas, 1993.

11. Malcolm, K.C., Poltrock, S. E., and Schuler, D. Industrial strength hypermedia: Requirements for a large engineering enterprise. In *Proceedings of Hypertext '91*, ACM Press, 1991, pp. 13–24.

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

13. Pearl, A. Sun's link service: A protocol for open linking. In *Proceedings of Hypertext'89* , ACM Press, 1989, pp. 137–146.

14. Rizk, A., and Sauter, L. Multicard: An open hypermedia system. In *Proceedings of ECHT'92*, ACM Press, 1992, pp. 4–10.

15. Schnase, J.L. HB2: A hyperbase management system for open, distributed hypermedia system architectures. Ph.D. Dissertation. Texas A&M University, College Station, Texas, 1992.

16. Stallman, R.M. Emacs: The extensible, customizable, self-documenting display editor. In *Interactive Programming Environments*, McGraw-Hill, 1984, pp. 300–325.

17. Whittlesey, J. Using Idraw, 1990. This document is part of the InterViews distribution available from Stanford University: "ftp://interviews.stanford.edu /pub/contrib/idraw.guide/".

18. Wiil, U.K. Extensibility in open, distributed hypertext systems. Ph.D. Dissertation. Aalborg University, Denmark, 1993. Available as Department of Computer Science Technical Report 93-2013, Aalborg University, 1993.

19. Wiil, U.K., and Leggett, J.J. Hyperform: Using extensibility to develop dynamic, open and distributed hypertext systems. In *Proceedings of ECHT'92*, ACM Press, 1992, pp. 251–261.

20. Wiil, U.K., and Leggett, J.J. Concurrency control in collaborative hypertext systems. In *Proceedings of Hypertext'93*, ACM Press, 1993, pp. 14–24.

21. Østerbye, K., and Wiil, U.K. The Flag taxonomy of open hypermedia systems. In *Proceedings of Hypertext'96*, ACM Press, 1996.