The Flag Taxonomy of Open Hypermedia Systems

Kasper Østerbye *Uffe Kock Wiil*

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

ABSTRACT

This paper presents a taxonomy for open hypermedia systems. The purpose of the Flag[1] taxonomy is manifold: (1) to provide a framework to classify and concisely describe individual systems, (2) to characterize what an *open* hypermedia system is, (3) to provide a framework for comparing different systems in a system independent way, and (4) to provide an overview of the design space of open hypermedia systems.

The Flag taxonomy builds on the achievements of the Dexter model. It extends the terminology of the Dexter model to adequately cover issues that relate to open hypermedia systems such as integration and use of third-party applications to edit and display hypermedia components.

Two of the most prominent open hypermedia systems, DeVise Hypermedia and Microcosm, are used as case studies. The Flag taxonomy is used to compare these systems on a carefully selected set of aspects that distinguish open hypermedia systems from other hypermedia systems.

KEYWORDS: Open hypermedia systems, Dexter model, taxonomy, link protocol, third-party viewers, integration

1 INTRODUCTION

The current trend in hypermedia systems design is towards open, extensible and distributed multiuser systems. In the past few years, several open hypermedia systems (OHSs) have been presented in the literature, including Sun's Link Service [20], Proxhy [16], Microcosm [4, 5, 15], Multicard [21], DeVise Hypermedia (DHM) [9, 10, 11, 12], Hyperform [24], SP3 [18], Chimera [2] and HyperDisco [26]. The fact that each of these OHSs have introduced their own hypermedia data model, architectural framework and link protocol (protocol for exchanging information with third-party applications) makes it very difficult to discuss and compare the different approaches in a system independent way.

Existing hypermedia reference models, such as the Dexter hypertext reference model [13], the Trellis hypertext reference model [8] and Lange's formal model of hypertext [17], do not adequately cover important aspects that distinguish OHSs from other hypermedia systems such as integration and use of third-party applications.

This paper presents the Flag taxonomy (in most places referred to as "the taxonomy"), which builds on the terminology of the Dexter 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 (e.g., Augment [7], Intermedia [19], KMS [1], Neptune [6] and NoteCards [14]). Even though the Dexter model pre-dates most OHSs, much of the Dexter terminology is still valid when discussing OHSs [10].

The main idea behind the taxonomy is to distinguish between storage aspects and runtime aspects on the one hand, and structure and contents on the other hand. This leads to four *functional modules* (FMs) and four *protocols* (see Figure 1). Each FM provides functionality to be used by its two neighbouring FMs through the available protocols.

The taxonomy provides a system independent framework for classifying, describing and comparing different OHSs. Since the taxonomy builds on the Dexter terminology, it can be used to contrast OHSs to other hypermedia systems. The taxonomy can be used at different levels of abstraction ranging from classifying hypermedia systems into broad categories to an in depth analysis

[1] The outline of the taxonomy resembles the Danish flag as well as the flag of other Scandinavian countries.

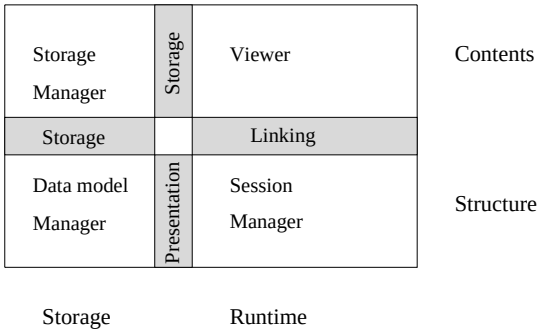


Figure 1: The Flag of hypermedia systems. It consists of four rectangles and a cross with four arms. Each rectangle represents a functional module of the system, and each arm represents the protocol between its two neighbouring modules.

of the individual FMs and protocols of the system.

The taxonomy relates to the Dexter model in the following manner. The *data model manager* corresponds to the Dexter storage layer and is responsible for storing the structure of a hypertext. Anchors are used as the "glue" between structure and contents. We also maintain the three component types in the data model: nodes, links and composites. However, the taxonomy is deliberately very open regarding the data model manager.

The explicit distinction on the runtime side between structure and contents and the division of contents into storage and runtime aspects, rearranges some Dexter concepts. The Dexter runtime layer defines two important concepts, instantiation (runtime presentation of a component) and session. An instantiation consists of three parts: a base instantiation (which represents the component), a sequence of link markers (which represent the anchors), and a mapping from link markers to anchors, *linkAnchor*. In the taxonomy, this mapping belongs to the *session manager module*, the base instantiation and link markers belong to the *viewer module*, and the session is the responsibility of the session manager module.

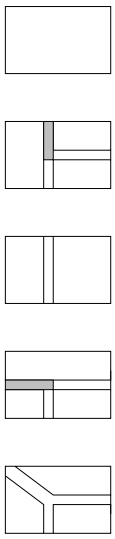
An important issue made explicit by the taxonomy, is that instantiations can be manipulated outside the structural part of the hypermedia system, which exposes the problems of integration and use of third-party viewers. We will use the term *viewer* to denote both applications that can edit and display components. The introduction of the viewer module has also allowed the Dexter within-component layer (corresponding to the *storage manager*) to be placed next to the Dexter runtime layer (in the form of the viewer module). This captures the important fact that most third-party viewers store their contents outside the hypermedia system.

The linking protocol is the runtime interface between the viewer and the session manager. We consider it one of the major contributions of the paper to discuss the required functionality of the linking protocol.

Section 2 shows how the taxonomy can be used to classify existing hypermedia systems into broad categories and to distinguish OHSs from other hypermedia systems. The taxonomy will be discussed in detail in Section 3. The focus is on runtime aspects: (1) the session manager module, (2) the viewer module, and (3) the linking protocol and its relation to the viewer and the session manager. To validate the usefulness of the taxonomy, we describe and contrast two prominent OHSs in Section 4 using the taxonomy. Section 5 concludes the paper.

2 HYPERMEDIA SYSTEM CATEGORIES

The taxonomy can be used to classify existing hypermedia systems into broad categories, by cutting the Flag in various ways (see Figure 2).



Monolithic
KMS, NoteCards

Hyperbase
Neptune, Sepia, EHTS

Embedded link
WWW

Link server
Sun's Link Service, Proxhy
Microcosm, Multicard, Chimera

Open hyperbase
DHM, SP3, HyperDisco

Figure 2: Different ways to slice a hypermedia system. Shaded protocols are not used.

The monolithic approach to hypermedia systems is characterized by having one module which is responsible for all aspects of the system. KMS and NoteCards belong to this category.

Hypermedia systems based on a hyperbase are characterized by a storage module which handles both contents and structure and a session manager (of varying sophistication) assisting viewers in maintaining contents and structure of the hypertext. The classic Neptune system and more recent hypermedia systems like Sepia [22] and EHTS [27] belong to this category.

The embedded link approach is a special case of the hyperbase approach with only two modules, a storage module and a runtime module. This approach does not explicitly distinguish between contents and structure. An example is the World-Wide Web (WWW) [3], with the storage part being WWW servers and the runtime part being WWW browsers (e.g., Mosaic and Netscape).

Link server based approaches are characterized by using third-party viewers to present *and* store the contents part of the hypertext. Link servers consist of a link base, which is responsible for storing the hypertext structure, and a session manager, which is responsible for assisting third-party viewers in maintaining structure. Sun's Link Service, Proxhy, Microcosm, Multicard and Chimera belong to this category.

The open hyperbase approach combines the hyperbase and link server approaches into an approach consisting of a storage module and a session manager. The storage module is responsible for storing the structure and is capable of storing the contents as well (if desired), or let the storage of contents be the responsibility of third-party viewers. The session manager is responsible for assisting third-party viewers in maintaining structure (and storing contents). DHM, SP3 and HyperDisco[2] belong to this category.

From a software system developer's point of view, all categories of hypermedia systems presented in this section (except the monolithic) can be considered open in the sense that there exist well-defined protocols between the different functional modules of the hypermedia system. This type of openness makes it possible to create new modules that adhere to the protocols (e.g., new WWW servers and browsers).

From a hypermedia system developer's point of view, only the link server and the open hyperbase approaches are considered open. The important matter in hypermedia systems is the distinction between structure and contents. The hyperbase and embedded link approaches impose a specific hypermedia data model on the viewers (specifying both structure and contents formats). In contrast, the link server and the open hyperbase approaches only impose a structure format on the viewers. Allowing viewers to store contents in different formats outside the hypermedia system is a basic requirement for integrating and using third-party viewers with the hypermedia system. Since the latter (more strict) definition of openness is preferred in this paper, the remaining sections will focus on systems belonging to the link server and open hyperbase categories.

[2] HyperDisco is based on Hyperform. Hyperform, being an open, extensible hypermedia system development environment, can be used to develop all categories of hypermedia systems.

Tailorability (and extensibility) is different from openness. Tailorability is a feature of individual modules (e.g., the session manager can be tailored to handle link resolution differently), while openness is a feature of the entire system. However, often openness depends on tailorability of the modules to allow for seamless integration of third-party viewers (e.g., by re-mapping the concepts from the viewer to the concepts of the session manager. Tailorability is further discussed in Section 3.

3 THE FLAG TAXONOMY

The primary goal of the taxonomy is to map out the design issues of OHSs and their most common solutions. The focus is on runtime modules (viewer and session manager) and the link protocol. Before describing the details of the taxonomy, general issues, which apply to all FMs and to all protocols of the Flag, will be presented.

Functional Modules. The following issues apply to all FMs regardless of their functional responsibilities:

- Tailorability. Each FM has functional responsibilities. To what extent can these be tailored and by whom?
- Each FM is bordered by two protocols. Do both protocols exist in a particular system, and if they do, can the FM handle one specific protocol, or can it handle different protocols?

Tailorability is useful in all modules of an OHS and can include many aspects. In this paper, tailorability is restricted to the issues summarized in Figure 3[3]. When observing the means to achieve tailorability in FMs, three broad categories of solutions should be considered.

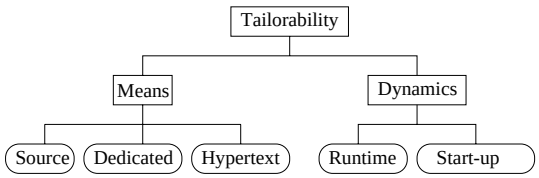


Figure 3: Aspects of tailorability for functional modules.

Source. Tailoring is performed at the source code level. This does not necessarily mean that the tailor has to understand all the details of the system. Source code level tailoring is typically used in connection with systems organized as a (possibly object-oriented) framework. DHM provides an extensible object-oriented framework for both data model and session manager.

Dedicated. Tailoring is performed using a dedicated mechanism. This can range from special-purpose languages, with an application programmers interface to

[3] In discussing issues and solutions, *taxonomic diagrams* like that of Figure 3 will be used. Angled boxes are issues, and rounded boxes are solutions.

operations in the FM, to graphical dialog boxes as known from many Macintosh and Windows applications. Microcosm uses a dedicated tool to manipulate the organization of link filters.

Hypertext. Tailoring is performed by creating hypertext structures, which are interpreted at runtime. The KMS action language use this rare solution (e.g., for its control structures).

Another aspect of tailorability is the dynamics of the customization. The distinction here is when the customization takes effect. Two possible solutions (among many) are runtime, meaning that the customizations take effect while the system is running, and start-up, which means that the system must be shut down and restarted for the customizations to take effect.

Protocols. Protocols have both a technical level, addressing how the two FMs next to the protocol exchange commands, and a contents level, defining the command repertoire of the protocol. The contents level of each protocol will be addressed later. Issues regarding the technical level are independent of the actual command repertoire:

- medium (e.g., DDE, Apple Events, TCP/IP sockets, internal library).
- format (e.g., *(send object message . args)*).

Normally, it is only the link protocol which is interesting at the technical level because most third-party viewers are not designed to use a specific (standard) communication protocol. This forces the session manager to support a range of different communication protocols. Davis *et al.* [1994] and Anderson *et al.* [1994] present a number of different ways to communicate with third-party viewers which are not designed to be integrated into a hypermedia system. The common solution is to build a wrapper for the third-party viewer, which transforms the command repertoire of the link protocol into whatever operations are available in the viewer. The use of wrappers is shown in Figure 4, where the wrapper is indicated as a thin line between the protocol and the FMs[4].

However, on some of the most common platforms there is currently a move towards standardized interprocess communication (OLE2 under Windows, and OpenDoc under Macintosh and Motif). One can therefore hope that the *technical* issues of protocols will soon cease to be an issue.

In the following discussions of FMs and protocols, the exact named parameterized operations provided from each FM will not be discussed in connection with the FM

[4]The extended Flag resembles the Norwegian flag, which to our great dismay gives a better description than the Danish flag.

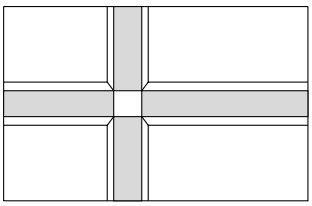


Figure 4: The extended Flag. The line along the protocols adapts the functional module to the specific protocol.

itself, but will be considered to lie within the protocol between the FM that provides the operation, and the FM that uses it. This allow us to discuss FMs in a more abstract manner and to talk about a given FM conforming to more than one protocol (specific set of operations).

3.1 Session Manager

The main responsibilities of the session manager are:

- Managing instantiations and tracking which viewers are responsible for presenting them.
- Resolving link activation, taking into account the current status of the session.
- Link availability, that is, controlling which links should appear as link markers in an instantiation.
- Serving as a mediator of messages in a collaborative setting.
- Coordinating the creation and maintenance of structure. This will be discussed together with the operations of the linking protocol.

Third-party viewers are capable of manipulating the contents of nodes outside the hypermedia system. This leads to two problems: (1) anchor consistency: how to maintain the linkAnchor mapping, and (2) contents location: how to deal with the situations that occur when node contents have been moved or (even worse) deleted from outside the hypermedia system.

Two types of anchors exist: positional and keyword. The anchor value of positional anchors specify a position within the node contents, while the anchor value of keyword anchors specify a generic part of the node contents rather than a position. Keyword anchors are easy to maintain and work quite well as demonstrated by local and generic links in Microcosm. Figure 5 summarizes some approaches to the anchor consistency problem for positional anchors. In DHM, the problem is identified as a special case of dangling links [10, case 4, page 43], but no attempts are made to handle the problem. There are two approaches to fully handle the problem. The first is to require the viewers to maintain the anchors, the second is to make different versions of the contents, at least ensuring that the anchors are correctly aligned to the

previous contents. However, versioning is rarely what is needed. HyperTED [23] applies heuristic methods to solving the anchor consistency problem. The main challenge of heuristic solutions is to be able to detect inconsistency, so the system can inform the end-user that the anchor is invalid.

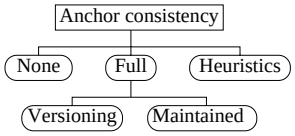


Figure 5: Anchor consistency issues for positional anchors.

The problem of node contents being moved to a different location from outside the hypermedia system has been addressed in HyperTED, which makes use of aliasing features of the Macintosh operating system to track renaming and moving of files within a single file system.

Ignoring the issue of anchor consistency, there is still room for interesting variations on how the resolver function is designed. Figure 6 summarizes the most important issues and solutions. The first sub-issue is how the link marker is resolved. Three solutions are presented: (1) a specific link marker is statically bound to a specific anchor, (2) the mapping from link marker to anchor is computed, and (3) to emphasize that there is a whole range of solutions to the resolution issue, the diagram includes a "once" solution, where the link marker mapping is computed the first time a link marker is looked up, and the mapping is hence static. When computing the link marker to anchor mapping as in both the computed and once solutions, it is an issue what input is used in the computation (including the actual link marker, end-user id, session history, phase of the moon, etc.).

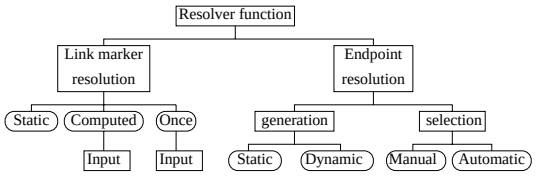


Figure 6: Resolver function issues.

Once the anchor has been found, the session manager can obtain the link attached to the anchor and must resolve the endpoints of the link. There are really two issues here. How to generate the set of endpoints for the link, and how to select which endpoints to present. The taxonomy distinguishes between static generation, which means that the endpoints are stored as part of the link, and dynamic generation, where the endpoints are computed. Selection has two possible solutions in the taxonomy, one is to let the end-user chose between all generated endpoints, the other is to let the system automatically select some.

There is often a correlation between the link marker and endpoint resolution issues, in that computed resolution often yields many endpoints, which makes it necessary to let the end-user decide which endpoint(s) to present. When the link marker is statically bound to a specific anchor and, therefore, to a specific link, it seems more appropriate to give the author control over which nodes should be presented, which implies that the system will automatically determine this (based on structural information given by the author).

There is also a correlation between anchor consistency and resolver function issues. As mentioned above, one way of addressing the consistency problem is to use keyword anchors. This often means that the keyword resolution is computed, at least the first time the keyword is looked up.

When a component is instantiated, it is important to consider which links are to be instantiated as link markers. Some links might be private annotations which should not be presented, or some links might require special end-user status to follow, or even see. It is the responsibility of the session manager to control which link markers are available. In the link protocol proposed in Section 3.3, there is an operation which will insert link markers into an instantiation, thus allowing the session manager to be in control of which anchors are to be available in the instantiation.

When not all anchors are included as link markers in an instantiation, it becomes difficult to maintain positional anchor consistency for the excluded anchors.

Tailorability is especially important in connection with the session manager, allowing flexibility in determining link availability and link resolution.

Finally, to support multiuser settings, the session manager must be able to receive notifications and determine the appropriate actions to take (if any) [11, 25]. For example, if the session manager is notified that a link has been added to a node, it must pass on the notification to viewers displaying this node, to inform them that link markers are no longer up-to-date.

3.2 Viewer

The main responsibilities of viewers are to present and manipulate the contents of hypermedia components. When assessing a viewer from an OHS's perspective, the following issues are central:

- Contents storage. Will the viewer store its contents by it self, or can it cooperate with the session manager to do so?
- Anchor handling. How and to what extent can the viewer maintain anchor values?
- Notification. To what extent can the viewer be used in a collaborative hypermedia setting?

There are different levels at which a viewer can support anchors. Figure 7 summarizes the anchor support issues and solutions. As indicated, possible solutions of directionality are to support source and destination anchors, which can either refer to values or objects within the document or refer to the document as a whole.

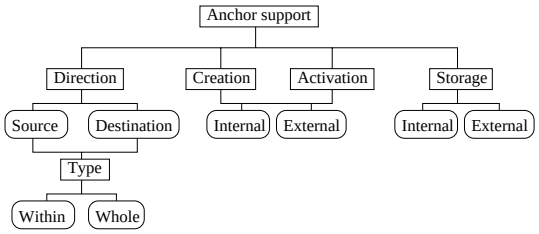


Figure 7: Aspects of viewers notion and awareness of anchors.

The issue of anchor creation deals with how the end-user creates an anchor, and the issue of activation deals with how the end-user activates an anchor. For both of these issues there are two possible solutions: (1) the viewer allows the end-user to initiate the create and activate commands inside the viewer (e.g., through a menu), or (2) the end-user will have to indicate the link marker and then issue the "activateLink" from the session manager.

Anchor storage can either be handled by the viewer itself, or by some mechanism outside the viewer. If the viewer is able to store its anchors, there is a good chance that anchors can be maintained in a consistent manner, even when the contents are edited outside the hypermedia system.

To support multiuser settings, viewers must be able to receive notifications and perform appropriate actions in response to them. For example, if a viewer is notified that its current instantiation is no longer up-to-date, it must somehow communicate this knowledge to the end-user. How this is done depends on the collaboration policies and capabilities of the viewer. Typical methods in a synchronous collaborative setting would be to automatically update the display or to inform the end-user that the displayed instantiation is outdated and a more recent version is available.

To illustrate the viewer issues, let us consider the usage of Microsoft Word for Windows as a third-party viewer. Word stores its contents in a proprietary format outside the hypermedia system. It can be notified as one can call Word commands from outside using Windows DDE. It can provide within-anchors for both source and destinations through the usage of the Word concept of "bookmarks". A bookmark is a named region of the text. The bookmark name can be used as the anchor value, and Word stores bookmarks internally in the document allowing the document to be safely edited outside the hypermedia system, as long as bookmarks are not deleted or renamed. Anchor creation and activation can be done internally, because menus can be tailored by means of the dedicated Word-basic language.

3.3 Linking Protocol

The main responsibility of the linking protocol is to provide the operations necessary to bind the viewer and session manager functionality together. The link protocol operations can typically be grouped into the following categories:

- Link activation. When the end-user activates a link, the viewer must inform the session manager of what link marker was activated.
- Spawning viewers. When a document is to be presented, the viewer and session manger must exchange document location, presentation information, and link marker information.
- Creating and maintaining structure. When the end-user, for instance, creates a new link, the viewer must inform the session manager of what link marker/selection was activated.
- Collaborative notifications. When the session manager receives notifications, these must be forwarded to the appropriate viewers.

Despite the taxonomic approach of this paper, we have found it useful to present a specific link protocol for illustrative purposes. The discussion of link protocol operations will take outset in the ten operations of the Dexter runtime layer: openSession, openComponents, presentComponent, followLink, newComponent, unPresent, editInstantiation, realizeEdits, deleteComponent and closeSession. In an OHS, the runtime functionality is divided between the viewer and the session manager. The Dexter model runtime operations will be revisited from this perspective, leading to the Flag link protocol (FLP) (Table 1).

The four operations openSession, closeSession, openComponents and presentComponent are not intended to be called from the viewer, and are therefore not part of the link protocol. The session manager will have its own user interface to enable opening and closing of sessions. The openComponents and presentComponent operations are internal operations used in particular by "activateLink" (followLink in Dexter).

Session manager operations	
activateLink	FLP version of Dexter followLink. Resolve link marker, and present destination component.
newComponent[1]	Opens a new instantiation on a newly created component (as in Dexter).
unInstantiate[1]	FLP version of Dexter unPresent. Removes instantiation from session.
saveEdits	FLP version of Dexter realizeEdits. Stores the contents of instantiation. Only necessary if the viewer cannot store its own contents.
deleteComponent[1]	As in Dexter, only we now call unInstantiate rather than Dexter unPresent.
setSelectedComponent	Set selected component to the component passed as parameter. The selected component is used by the following two operations.
createSpecifier	Add specifier to *selected* link based on the link marker passed as parameter.
addToComponent	Add component passed as parameter to *selected* composite.
Viewer operations	
presentInstantiation	Retrieves and presents an instantiation.
presentAnchors	Creates link markers for a set of anchors.
gotoAnchor	Indicate an anchor as the destination in an activateLink operation.
saveInstantiation[2]	Saves the changes done to an instantiation.
closeInstantiation[2]	Closes the view of an instantiation.
markAsObsolete	Marks the contents in some clear way to indicate that the stored version is now newer than the one presented here. Specific implementations can do anything from deleting the contents, to making it read-only or to change background color.

Table 1: Flag link protocol.

Notes: 1) If viewers store their own contents, these functions merely notifies the session manager that the action took place in the viewer. 2) These operations are only needed if the session manager wants to implement secure closing of sessions, otherwise saving and closing is initiated by the end-user directly in the viewer.

When presenting a hypermedia component (e.g., when using activateLink) the presentation specification includes information on what viewer should be invoked. Most viewers can be invoked with a description of what component to present. Alternatively, a running viewer can be instructed to present a specific component using the viewer operation "presentInstantiation". Part of instantiating a component is to visualize (a set of) its anchors as link markers in the viewer. To do this, the session manager call the viewer operation "presentAnchors". To indicate the destination of a link in a document, the viewer operation "gotoAnchor" indicates the destination link marker based on an anchor value.

The editInstantiation operation becomes part of the viewer functionality and as such does not belong to the link protocol. The realizeEdits operation saves the changes done to a given instantiation. Most third-party viewers will save their own contents – however, in the open hyperbase approach, the session manager should provide a way for a viewer to store contents; in the FLP the "saveEdits" operation replaces the realizeEdits operation.

Closing an instantiation, as in unPresent and deleteComponent, is now done by the viewers themselves. However, to enable the session manager to track the set of instantiations, the viewers must inform the session manager when an instantiation is closed or deleted. To emphasize the change of initiation, the Dexter unPresent operation is renamed unInstantiate.

Though it is not strictly necessary, the FLP specifies that the session manager must be able to instruct the viewers to save and close the instantiation. The Dexter closeSession specifies that closing the session can result in loss of changes. The designer of the session manager should be given the choice to implement a more acceptable behaviour.

To enable structural editing, there is a need for an operation to add new specifiers to a link, and to include components into a composite. While this can be done in several ways, the FLP specifies a simple protocol for doing this. The session manager maintains a *selected component*, which can be set by the setSelectedComponent operation. To create a link between two nodes, the newComponent operation creates a new link. The new component is automatically made the selected component. Then a node is located, and a link marker is created. The createSpecifier is then called with the link marker as argument, and a new specifier is created in the selected link. This is repeated for other endpoints of the link. To add a component to a composite is quite similar. The composite is made the selected component, and the addToComponent is issued from the viewer of an existing instantiation.

In synchronous collaborative hypermedia systems, the session manager must be able to track changes to the hypertext (made by other end-users), and notify the end-user of such changes. The FLP specifies two simple ways to let the session manager address the viewer: "presentInstantiation" followed by "presentAnchors" can be used to instruct the viewer to reload the contents and present the current set of anchors, and "markAsObsolete" can be used to inform the viewer that the instantiation is no longer current.

3.4 Other Functional Modules and Protocols

Since the main emphasis of this paper has been on runtime aspects of the Flag, this section will only briefly describe the remaining modules and protocols.

Data Model Manager. The Dexter model gives a rather specific description of the fundamental data model, detailing how components are to be realized, at least functionally. Since the Dexter model has been published, implementation based on the original model has revealed some problems and open ends in the original specification [10, 11]. New models such as the link server approach in Microcosm are using a very different data model than proposed by the Dexter model. Experiences with DHM and Hyperform have emphasized the importance of having a general hypermedia data model framework which can be extended and tailored towards the specific needs of individual applications and application areas.

Storage Manager. Even though the taxonomy does not state anything specific about the data model, there is a fundamental insight in separating the storage into structure and contents modules. The issue of contents storage format pertains to the storage manager. If several viewers agree on a storage format (e.g., HTML), they can share the same document contents, with some allowing editing and others being passive viewers.

Presentation Protocol. This protocol defines the data model operations available to the session manager, and it defines the session manager call-back operations available to the data model manager (e.g., in relation to cooperative events).

Storage Protocol. The storage protocol encapsulates the storage manager from both the viewer and the data model manager. There is an important point to be made here in relation to OHSs. While the storage manager module can provide storage to third-party viewers, it can also serve as information provider by delivering contents in a format which can be interpreted by existing viewers (e.g., ASCII, RTF, Postscript, etc.). In the extended Flag, the storage protocol has two places where one can put a wrapper. If the wrapper is placed on the data model manager (or viewer) side, it can be used to introduce virtual (computed) contents. In this way various kinds of read-only information can be incorporated into the hypermedia system (e.g., UNIX manual pages).

4 CASE STUDIES

To illustrate the differences between the two categories of OHSs, "open hyperbase" and "link server", we will describe and compare DHM and Microcosm using the taxonomic aspects presented in the previous section.

4.1 Open Hyperbase: DHM

The DHM system is described by Grønbæk and others in [9, 10, 11, 12]. The first reference is particularly interesting from the perspective of OHSs, since it describes in some detail how third-party viewers can be added to the system.

The data model manager and the session manager (both based directly on the Dexter model) can be tailored in the form of source code level customization, which takes effect at runtime. This is done using an interpreter and an object-oriented framework for the system.

Contents storage is either entirely the responsibility of an object-oriented database, or node contents can be handled by the viewers (in DHM terminology called editors). The link protocol is internal to the session manager. Integration of third-party viewers are done using wrappers on the viewer side of the link protocol.

Regarding anchor consistency, the DHM system either depends on the viewer to do this, or there is no support for this. The resolver function is flexible, as all solutions are possible through tailoring. The default behaviour is that the link marker resolution is static, and the endpoint selection is automatic (bring up all endpoints of a link).

The DHM system is capable of handling viewers with all kinds of anchor support (cf. Figure 7). The link protocol has three levels, for fully open editors, semi-open editors and closed editors. These levels reflect that the system is able to handle viewers with different levels of anchor awareness. It is not clear if the session manager provide external anchor creation and activation in those cases where the viewer is not able to do it by itself.

The link availability issue is not directly addressed in any paper we know of, but the session manager should be able to restrict which anchors are instantiated into link markers, through tailoring of the instantiation concept in the session manager.

The DHM system provides support for collaborative work in both the data model manager and the session manager. The notification mechanism allows different end-users to be informed of changes done by other end-users, and the locking mechanism allows collaboration in a se-

cure fashion.

4.2 Link Server: Microcosm

Microcosm is described in [4, 5, 15]. Microcosm is based on a link-filter approach [15], which basically is a special way of organizing the link marker resolver function, merging static, once and computed. Endpoint selection is manual.

Link marker resolution can be tailored using a dedicated tool to manipulate the order in which link markers are examined in the so called filter-chain. Adding new filter types require source code level changes using a special application programmers interface.

The anchor consistency issue is to a large extent solved by extensive usage of keyword anchors, but the system supports viewer maintained positional anchors as well. The link protocol is based on a fixed command repertoire, but the flexibility of the resolver function makes it de facto possible to have different levels of the link protocol. The link protocol consists of a set of Windows DDE calls. Integration of third-party viewers are done using a number of different methods [5].

Like DHM, Microcosm will handle viewers with all kinds of anchor support. As Microcosm is a link server approach, viewers are always responsible for storing their own contents. The data model is very simple and does not allow further specialization, for example, one cannot easily create typed nodes and links.

The link-filter approach used by Microcosm directly addresses the issue of link availability, in that the filter chain can be used to filter out those links that should not be part of a specific instantiation.

In its present form, Microcosm does not address collaborative issues.

4.3 Comparison

The main difference between DHM and Microcosm is that DHM allows the data model to be tailored, while Microcosm does not. DHM allows the choice between storing the node contents within the system, or externally, maintained by the viewer, where Microcosm requires the viewer to store it. DHM has its link protocol internal to the session manager, whereas the link protocol of Microcosm is specified in the form of an interprocess communication protocol. This means, to adapt a new viewer to the system, one has to specialize the internals of DHM, whereas one does not have to do this in Microcosm. The difference is illustrated in Figure 8, where the dotted line indicates where the interprocess communication takes place.

Microcosm is built around the resolver function, and the link marker resolution of Microcosm seems to be more readily tailorable than that of DHM. The link-filter approach and the extensive usage of keyword anchors gives Microcosm better mechanisms for flexible control over link availability, though the issue can be addressed in the DHM architecture. DHM supports collaborative work through notification and lock mechanisms, where Microcosm gives no support at all.

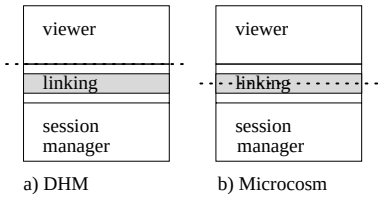


Figure 8: Two approaches to link protocols. a) as done in the DHM system, and b) as done in the Microcosm system. The dotted line indicates where the modules are split into separate processes.

It seems like Microcosm is more suitable for integrating third-party viewers, because the system is better geared towards it. On the other hand, the DHM system provides a more suitable platform for writing new viewers, because the data model can be tailored to better suit the needs of new viewers.

5 CONCLUSION

The Flag taxonomy provides a framework which allow us to: (1) classify existing hypermedia systems, (2) characterize what an *open* hypermedia system is, and (3) examine (describe and compare) OHSs independent of the particularities of specific systems. The taxonomy also allowed us to differentiate between important features such as openness and tailorability. Within the framework of the Flag, we presented a number of issues and possible solutions, in particular for the session manager and viewer modules, which can serve as the basis for a more detailed description of existing OHSs and as a starting point for design of new OHSs. The taxonomic diagrams will serve as check lists to help getting started on a specific design.

In Section 2, we used the taxonomy to classify existing hypermedia systems into broad categories. However, broad classification will often ignore interesting details. For example, the WWW was characterized as a closed hypermedia system, because it does not have a linking protocol. However, WWW should not be dismissed so easily. The WWW consists of two main functional modules, a server and a browser. The server sends HTML encoded information on request from the browser, and the browser then displays this information. But this is only the case when the URL is “http”. If the URL specifies that a “file” is to be retrieved, both the Mosaic and the Netscape browser has a file type mapping

which specifies what external application to use for presenting the file. It is thus more fair to say that these browsers serve as session managers, and the external applications serve as viewers. However, the linking protocol is very simple, with viewers supporting only the *presentInstantiation* operation. The Netscape browser supports a vide range of operations which can be called from the viewers, giving a rich linking protocol towards the session manager.

In Section 4, we presented a description and comparison of two existing OHSs, DHM and Microcosm. The descriptions build strongly on the vocabulary established in the taxonomy. The fact that none of these descriptions are very long, indicates that the taxonomy provides a framework to understand and evaluate the general characteristics of OHSs with minimal effort. There are of course more subtle aspects of these systems which can not be captured by describing the systems according to the taxonomy. It serves as a check list to ensure that one has indeed covered the necessary aspects of an OHS.

One of the current trends in application development is interoperability, allowing one application to draw on the powers of other applications. We have given an example link protocol, which provides insight into what kind of functionality third-party viewers must support to be integrated into a hypermedia system. Viewers must be controllable from the session manager, which is becoming the norm with systems such as DDE and OLE2 under Microsoft Windows, and Apple events under the Macintosh operating system. Viewers must also be able to support anchors in one form or the other (e.g., bookmarks in Microsoft Word, or named cells in Excel). Finally, viewers need to be tailorable to allow anchor operations to be initiated conveniently from within the viewer.

The taxonomy can serve as the basis for further development into an actual reference model for OHSs by settling on a specific data model (or an extensible framework) and a framework for the session manager, and by specifying the unspecified protocols.

ACKNOWLEDGEMENTS

The participants at the ECHT '94 Workshop on Open Hypermedia Systems have all been influencing this work. We are especially thankful to Serge Demeyer and David Hicks who commented on an earlier draft of this paper.

REFERENCES

1. Akscyn, R.M., McCracken, D.L., and Yoder, E.A. KMS: A distributed hypermedia system for managing knowledge in organizations. *Commun. ACM*, 31, 7 (July 1988), 820–835.

2. 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.

3. Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H.F., and Secret, A. The World-Wide Web. *Commun. ACM*, 37, 8 (Aug. 1994), 76–82.

4. 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.

5. 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.

6. Delisle, N., and Schwartz, M. Neptune: A hypertext system for CAD applications. In *Proceedings of SIGMOD'86*, ACM Press, 1986, pp. 132–143.

7. Engelbart, D.C. Authorship provisions in AUGMENT. In *Proceedings of the COMPCON'84*, 1984, pp. 465–472.

8. Furuta, R., and Stotts, P.D. The Trellis hypertext reference model. In *Proceedings of the NIST Hypertext Standardization Workshop*, 1990, pp. 83–93.

9. Grønbæk, K., and Malhotra, J. Building tailorable hypermedia systems: The embedded-interpreter approach. In *Proceedings of OOPSLA'94*, ACM Press, 1994, pp. 85–101.

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

11. 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.

12. Grønbæk, K. Composites in a Dexter-based hypermedia framework. In *Proceedings of ECHT'94*, ACM Press, 1994, pp. 59–69.

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

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

15. Hill, G., Wilkins, R., and Hall, W. Open and reconfigurable hypermedia systems: A filter-based model. *Hypermedia*, 5, 2 (1993), 103–118.

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

17. Lange, D.B. A formal model of hypertext. In *Proceedings of the NIST Hypertext Standardization Workshop*, 1990, pp. 145–166.

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

19. Meyrowitz, N. Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework. In *Proceedings of OOPSLA'86*, ACM Press, 1986, pp. 186–201.

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

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

22. Streitz, N., Haake, J., Hannemann, J., Lempke, A., Schuler, W., Schütt, H., and Thüring, M. SEPIA: A cooperative hypermedia authoring environment. In *Proceedings of ECHT'92*, ACM Press, 1992, pp. 11–22.

23. Vanzyl, A. Open hypermedia systems - comparisons and suggestions for implementation strategies. In *Proceedings of the ECHT'94 Workshop on Open Hypermedia Systems*. Dept. of Computer Science Technical Report R-94-2038. Aalborg Univ., Fr. Bajers Vej 7E, 9220 Aalborg Ø, Denmark, 1994, pp. 11–15.

24. 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.

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

26. Wiil, U.K., and Leggett, J.J. The HyperDisco approach to open hypermedia systems. In *Proceedings of Hypertext'96*, ACM Press, 1996.

27. Wiil, U.K. Issues in the design of EHTS: A multiuser hypertext system for collaboration. In *Proceedings of HICSS-25*, IEEE Computer Society Press, 1992, pp. 629–639.