Vrije Universiteit Brussel
Faculteit Wetenschappen

Ending the Tyranny of the Link: Adding Paths to the Dexter-model

Serge Demeyer

Techreport vub-prog-tr-95-07

Programming Technology Lab
PROG(WE)
VUB
Pleinlaan 2
1050 Brussel
BELGIUM

Fax: (+32) 2-629-3525
Tel: (+32) 2-629-3308
Anon. FTP: progftp.vub.ac.be
WWW: progwww.vub.ac.be

Ending the Tyranny of the Link: Adding Paths to the Dexter-model

Serge Demeyer
Vrije Universiteit Brussel / Faculty of Sciences
Programming Technology Lab (PROG)
Pleinlaan 2
B-1050 Brussels, Belgium
phone: (+32) 2 629 34 91
e-mail: sademeye@vnet3.vub.ac.be http://progwww.vub.ac.be/

ABSTRACT

Hypermedia technology is a potential benefit for all computer applications that deal with information. To penetrate new markets, hypermedia systems should be tailorable to specific application domains. We claim that an open, extensible hypermedia system is crucial to attain such tailorability. A hypermedia system should be capable to integrate 1) facilities for incorporating vendor-independent document viewers and 2) flexible linking facilities that access external information repositories.

This document describes how we extended the Dexter model with the "path" concept, to model hypermedia systems with extensible link engines. We show that paths absorb the notion of links and make it possible to integrate various strategies for resolving links. This proves our claim that "paths end the tyranny of the link". This is demonstrated with a case from the Software Engineering Community: a framework browser. The case involves a hypermedia system that integrates a home-cooked world-wide web browser, an off-the-shelf word processor (Microsoft Word) and a programming environment for Smalltalk (VisualWorks). Besides interpreting embedded (HTML style) anchors, the system is able to query the Smalltalk environment to link documentation to Smalltalk source code. The case serves as a proof of concept that an extensible hypermedia system can penetrate specific application domains.

Keywords: Hypermedia, Open hypermedia systems, Dexter, Zypher, Software Engineering, Object-oriented framework, Design pattern, Framework browser.

1. INTRODUCTION

Within the hypermedia community lives a bizarre attitude towards the link concept. In hypermedia, links are the means to relate distinct chunks of data and thus essential for shaping the complex information structures envisioned in the field. But in most hypermedia systems, links are very static in nature. This explains pejorative statements like "link = goto", "linking considered harmful", "ending the tyranny of the link", …. We observe a mismatch between the static link structures of today's hypermedia systems and the dynamic information structures in specific application domains. Although hypermedia technology is a potential benefit for all computer applications that deal with information, the static link structures hinder the adoption of such hypermedia technology in domain-specific applications. To solve this, links should become a dynamic element of hypermedia systems, exploiting knowledge inside various information repositories for presenting dynamic information structures.

The Dexter Hypertext Reference model [Halasz,Schwartz90] is a de facto standard serving as the basis for several of today's hypermedia systems. As such, it is sometimes seen as the typical example of the current generation of monolithic hypermedia systems. One of the known deficiencies of the model is precisely its limited capabilities to express dynamic link behaviour. The Dexter model should be equipped with an *extensible link engine*.

We add the *path* concept to the Dexter model in order to model hypermedia systems with extensible link engines. Paths absorb the notion of links and make it possible to integrate various strategies for resolving links. The approach is demonstrated with an experimental hypermedia system called *Zypher*[1]. Zypher integrates a home-cooked world-wide web browser, an off-the-shelf word processor (Microsoft Word) and a programming environment for Smalltalk (VisualWorks). Besides interpreting embedded (HTML style) anchors, the system is able to query the Smalltalk environment to link documentation to Smalltalk

[1] The name Zypher stems from the Louis Zypher character performed by Robert De Niro in the movie "Angel Heart".

source code. Such a hypermedia system is an ideal vehicle to document an object-oriented framework as the system allows one to look at the source-code from various angles, guided by the documentation. That is why we call such a software artefact a *framework browser*.

1.1. Document Organisation

Section 2 defines some terminology. Monolithic hypermedia systems are contrasted with open hypermedia systems by studying link facilities and third-party application integration. Section 3 gives a short overview of the original Dexter specification and proposes some modifications to turn the monolithic Dexter model into an open one. Basically, we rearrange some Dexter concepts to cope with the problems of third-party application integration and introduce the path concept to handle special link behaviour. Section 4 demonstrates the possibilities of the revised Dexter model by means of an implementation called Zypher. This implementation is used to construct a framework browser. Section 5 describes related work in the hypermedia and the software engineering community. Section 6 restates the importance of an extensible link engine. Finally, section 7 makes concluding remarks.

2. TERMINOLOGY

2.1. Monolithic vs. Open Hypermedia Systems

One of the original goals of the *Dexter Hypertext Reference model* [Halasz,Schwartz90] was to define a hypermedia interchange standard for a wide range of existing and future hypermedia systems. The model attempted to capture the state of the art of that time's most prominent hypermedia systems. As such, the Dexter model had a considerable impact on all later hypermedia systems; some hypermedia systems even initiated their design using the Dexter specification[2]. Nowadays, the Dexter model is regarded as the representative for the generation of *monolithic hypermedia systems*. Several authors (see [Halasz91], [Legget,Schnase94]) pointed out that monolithic systems lack important functionality that prevent hypermedia from finding widespread use. The emerging generation of *open hypermedia systems* (OHS) [Wiil,Østerbye94] tries add this lacking functionality and the work described here is part of that joint effort.

This document focusses on two particular issues in open hypermedia research, namely extensible link engines and third-party application integration. Several examples from existing hypermedia systems will be used to qualify the discussion. Related work will be described in section 5.

2.2. Static and Dynamic Links

One of the issues in the area of OHS is to "end the tyranny of the link" [Halasz91]. Besides traversing static graph-structures, hypermedia systems should be able to navigate non-graph structures as well a virtual graph structures. Typical examples for non-graph structures are the set-based model of Parunak [Parunak91] (links depend on a topological ordering of nodes using sets) and the spatial layouts of Aquanet [Marshall,Shipman93] (links are derived from the relative positions nodes take in a two-dimensional layout). Prototypes for virtual structures can be found in many settings. We name but a few: Trellis [Stotts91] (links are based on the computational model of petri-nets), QRL [CharoentkitkarnEtAl93] (links are based on fast string searches making use of full-text indexing techniques) and MacWeb [Nanard,Nanard93] (links jump to synthesised virtual documents: documents that are created at navigation-time). In this document, we will use the name *dynamic links* to refer to the hypermedia components that relate non-graph structures and virtual graph structures. The adjective dynamic stresses the fact that the endpoints of the link are produced by some computation. This is in contrast with the traditional *static links*, where the end-points of the link remain constant once the link is created.

We observe static and dynamic links in the popular world-wide web (WWW) [Berners-LeeEtAl94] as well. In most occasions, HTML links represent static graph structures used for simple navigation: nearly all anchors embedded in a HTML document contain a fixed reference to another document. Activating such an anchor brings up the referenced document, sometimes with a target area selected. But an anchor can also trigger a 'computation' on the WWW server that returns a HTML document. Such anchors are part of a virtual structure and represent dynamic links. Other examples for dynamic links can be found in WWW servers that construct HTML documents on the fly [DuvalEtAl95].

[2] Notably DeVise hypermedia-DHM [Grønbaek,Trigg94] and the Amsterdam Hypermedia Model-AHM [Hardman,Bulterman,VanRossum94].

The evolution of static links to dynamic ones calls for hypermedia systems that elevate link behaviour so that different link resolving techniques can be exploited. That is precisely the aim of our work: to build a hypermedia system with an *extensible link engine*. We extend the Dexter-model with *paths*, that are responsible for the management of links. Paths are equipped with a set of named *resolvers* so that link functionality can be encapsulated in separate plug-and-play modules. The default behaviour supports static navigational links, but plugging extra resolvers extends the systems link behaviour. Examples of powerful resolvers are described in section 4.

2.3. The Interoperability Problem

Another important subfield in OHS research investigates how to integrate third-party applications as document presenter programs. In the OHS field this problem is referred to as the *interoperability problem*. When integrating third-party applications, hypermedia systems tend to dissolve in the background of the operating system, playing the role of so-called *link services*. Hypermedia link services have much in common with the clipboard concept in recent desktop environments like System 7, Microsoft Windows and OS/2. Users are free to use their favourite applications (word processors, drawing packages, spreadsheets, mail applications, etc.) to manipulate information and just like they can transfer data to and from the clipboard (by issue a command like cut/copy/paste), they can register and activate links in the background link service (by issue a command like create anchor /follow link/…). The hypermedia system communicates with the corresponding applications to produce the desired effect.

In OHS the term *viewer* is used to refer to an application that presents documents to the user. This is a misleading term as the application allows users to modify the contents of the document as well. Nevertheless, in the rest of the paper we will adopt this terminology when referring to an application that presents and edits documents as part of the overall hypermedia functionality.

Examples of third-party application integration can be found in Microcosm [Davis,Knight,Hall94], Proxhy [Kacmar,Leggett91] and Chimera [Anderson,Taylor,Whitehead94]. Most world-wide web browsers offer limited facilities for third-party application integration as well. Applications like Mosaic and Netscape manage a list of external applications that can be called to handle special document formats.

Third-party application integration is currently a hot topic in OHS research. It is a promising subfield (certainly in the light of emerging interprocess standards like OLE and OpenDoc) and in the following sections we will elaborate on the interoperability problem. Nevertheless, we would like to emphasise the importance of extensible link engines. Hypermedia systems deal with personalised information presentation [Ashman94], so it is important how one can integrate information viewers of the user's choice (and not of the system's choice). But the way the information is organised is at least as important, which means that Hypermedia systems should be able to interpret the information structures of the user's choice as well. We propose an extensible link engine as the instrument to build and inspect personalised information structures: it allows a user to express and examine relations between information-chunks the way he or she wants, not the way the hypermedia system dictates it. In sections 4.3 and 4.4 we will show how we reached that goal.

3. VIEWING THE DEXTER DESIGN WITH OPEN EYES

The previous section focused on two important issues in open hypermedia research: extensible link engines and interoperability. Now we are going to describe how we turned a monolithic hypermedia model into an open one. We start with a brief discussion on the original design of the Dexter model and then rearrange the Dexter concepts to cope with the interoperability problem. Afterwards, we will introduce the path concept to enable the integration of different link resolving strategies and attain the desired goal: an extensible link engine.

3.1. The Original Dexter Design

The design of the Dexter model aimed to solve well-known problems for the generation of monolithic hypermedia systems. In short these problems are

- How to encapsulate the format used to describe the information inside the building blocks of the hypermedia network ?
- What are the best mechanisms to structure the hypermedia network itself ?
- How can one access, manipulate and view the information inside the hypermedia network ?

Dexter addressed these problems with a 3-layer architecture (within layer, storage layer, run-time layer), each one responsible for managing a segment of the problem space. Each layer consists of a set of concepts that serve as building blocks to model the hypermedia. Concepts exist that model the transition from one layer to the other as well. We will briefly discuss the concepts that matter to the rest of the document; for an extensive description of the Dexter model we refer to the original paper [Halasz,Schwartz90].
The within layer does not comtain concepts relevant in the following discussion. The storage layer defines:

- *atomic components*: chunks of information; in traditional hypermedia terminology this corresponds to nodes
- *anchors*: a reference to a substructure of a component
- *link components*: a collection of references to other components; a reference includes a component specifier, an anchor (identifier) and a traversal direction
- *presentation specifications*: primitive values in an unspecified format specifying how the components should be presented to the user
- *component specifiers*: primitive values in an unspecified format
- *resolver-function:* is able to translate a component specifier in a component

The run-time layer defines:

- *instantiations*: the run-time equivalents of components
- *markers*: the run-time equivalents of anchors.

It is interesting to see how the transition from one layer to another is accomplished. The Dexter model defines two concepts: *hypertext* (representing the storage layer) and *session* (representing the run-time layer), that serve as repositories for all the objects in the corresponding layer. More importantly, these concepts model the transition from the lower layer to the layer they represent. In other words, the hypertext concept knows how to map contents on components (anchor values on anchors) and knows how to store and retrieve the information inside components (anchors). The session concept knows how to map instantiations and markers on components and anchors, display and edit instantiations (by an interpretation of the presentation specifications associated with the corresponding component) and display and manipulate the hot spots inside the instantiation (based on the markers associated with the departing links).

Another matter that needs some attention is Dexter's link semantics. When an application activates a marker, the corresponding anchor is used to look up the actual link. The directions associated with the different end-points of the link are applied to filter out the target end-points that will be presented to the user. Such end-points consist of anchors, components and presentation specifications. The components and anchors are retrieved by the storage layer and mapped on instantiations and markers with the run-time layer. Finally these run-time components are presented to the user, depending on the presentation specifications found in the corresponding end-point.

We finish this overview with an examination of dynamic links in the Dexter model. According to the original specification, each endpoint of a link contains a component specifier, which is an indirect reference to a component. Each component specifier is passed through a resolver-function (defined on the hypertext and session concept) that returns the actual component. For static links such a component specifier contains the globally unique identifier of the component it refers to. To provide for dynamic links one constructs endpoints using "computed" component specifiers and makes sure the resolver-function is able to resolve these specifications.

3.2. Dexter and the Interoperability Problem

As noted in [Legget,Schnase94], the Dexter model aimed at a closed system architecture: Dexter implicitly assumes that the applications making up the within and run-time layers are known in advance; Dexter wants all the hypertext structures to be modelled inside the storage layer and supposes that all components live in one and only one hypertext and that all anchors are valid in all hypertexts. These assumptions are invalid for open hypermedia systems.

On the other hand, open hypermedia systems should at least tackle the same problems as closed systems, so it seems natural to build on the accumulated knowledge of the Dexter model. Moreover, as noted in [Grønbaek,Trigg94], the Dexter design may help in solving the interoperability problem. A hypermedia system that maintains links between documents managed by third-party applications needs some way to refer to substructures of such documents, irrespective of the actual contents of these documents. That is precisely the role of the anchors in the Dexter model: mediating between the actual reference inside components and the way this is represented in the storage layer. In Dexter, an anchor is an object having an identifier and a value. As far as the hypermedia system concerns it manipulates complete anchors and anchor identifiers; applications dealing with the within layer (i.e. the third-party document viewers) interpret the anchor values to defer the actual substructure the anchor refers to. In much the same way, presentation specifications were necessary in Dexter to capture device dependent display information. The hypermedia system is able to attach presentation specifications to the objects it handles; the actual applications are able to interpret those specifications and turn them into actual presentations.

So Dexter provides the means to model hypermedia structures managed by third-party applications. But to model an actual OHS, Dexter lacks a way to refer to the external viewers themselves. Hypermedia systems like Microcosm [Davis,Knight,Hall94] and Proxhy [Kacmar,Leggett91] furnish some kind of interpreters mediating between the hypermedia system and the external applications. Following the Dexter philosophy, it seems natural to attach those interpreters to the transition concepts hypertext and session.

We adapt the Dexter model to deal with such interpreters: a set of named loaders is added to the hypertext concept and a set of named editors to the session concept. Each *loader* mediates between the hypermedia system and external applications that are able to store and retrieve the contents of storage layer objects from sources outside the hypermedia system (file system, databases, …). An *editor* translates the contents of the components into a form that can be manipulated by external applications. The question rises how the model is able to know which loader/editor to call for a certain object. For this purpose, we added a *tag* attribute to all objects in the storage and run-time layer. This tag is used to look up the named loader/editor in the corresponding layer. Tags are very important to enable run-time extensions of hypermedia systems.

A complete protocol between the concepts in the different layers is defined, but it is beyond the scope of this document to go into the details. We refer to section 4.2 for examples that illustrate how this architecture is able to integrate third-party applications as document viewers.

3.3. Incorporating an Extensible Link Engine

Amongst others, [Legget,Schnase94], [Grønbaek,Trigg94], [Grønbaek,Hem,Madsen94], [Grønbaek94], [Garzotto,Mainetti,Paoline94] have criticised the original Dexter model, because of its limited capabilities to manipulate dynamic structures based on information residing in external repositories.

Surely the model provides computational links, but the computations are bound to obtain there input from the storage or the run-time layer. Computational links can express queries like "Return all components that are linked to a component updated today" because the query system can inspect the hypermedia structure (the hypermedia network is defined in the storage layer) and attributes containing information like the last modification date. Things get more complicated when we need a computational link returning all the components mentioning the word "hypermedia". This requires the resolver-function (part of the storage layer) to be expressed in terms of within-layer concepts, which violates the layering principle of the Dexter model.

But even if one tolerates a resolver-function that is not Dexter compliant, there are fundamental shortcomings with respect to dynamic structures. The least of the problems is that it is not determined how to extend the link engine: the Dexter specification does not say how one can construct special purpose component-specifiers and adapt the resolver function accordingly. Worse is that the resolver-function is designed to handle the simplest forms of dynamic structures (i.e. links where one end-point is fixed and the other one is computed). Links based on searching (like the "generic links" in Microcosm [DavisEtAl92]) fit in the resolver-function model, but it is almost impossible to shape non-graph structures like the set-based model of Parunak [Parunak91] or the spatial layouts of Aquanet [Marshall,Shipman93]. Not to mention applications like the Geographical Information Systems envisioned in [Bieber,Kacmar95].

The intrinsic problem of the Dexter model (and all other link-based hypermedia systems) is that it sticks to the notion of links to build information structures resembling graphs. But most information structures are maintained outside hypermedia systems and many do not fit well in a graph. This means that people who want to incorporate external information structures in a link based hypermedia system —i.e. people that tailor a hypermedia system to a particular application domain— will encounter problems.

We propose an extensible link engine as a solution. An extensible link engine is free to chose the linking model that best suits the needs of the information structure it represents. To incorporate an extensible link engine into the Dexter model, we add a fourth linking layer to the Dexter model that contains one concept equivalent to the hypertext and session concepts: *paths*. Paths are equipped with a set of named resolvers and each resolver implements one link resolving strategy. The final model is depicted in figure 1. The linking semantics of this extended model is as follows. When a marker is activated on an instantiation, the action is passed to the path with the corresponding component and anchor. With every anchor a linking tag is associated, identifying the resolver that is supposed to handle the requested link activation. The resolver does whatever it finds appropriate, but is supposed to return control to the hypermedia system with the protocols defined on the run-time layer. The resolver is allowed to access the resources in the storage and the within layer[3].

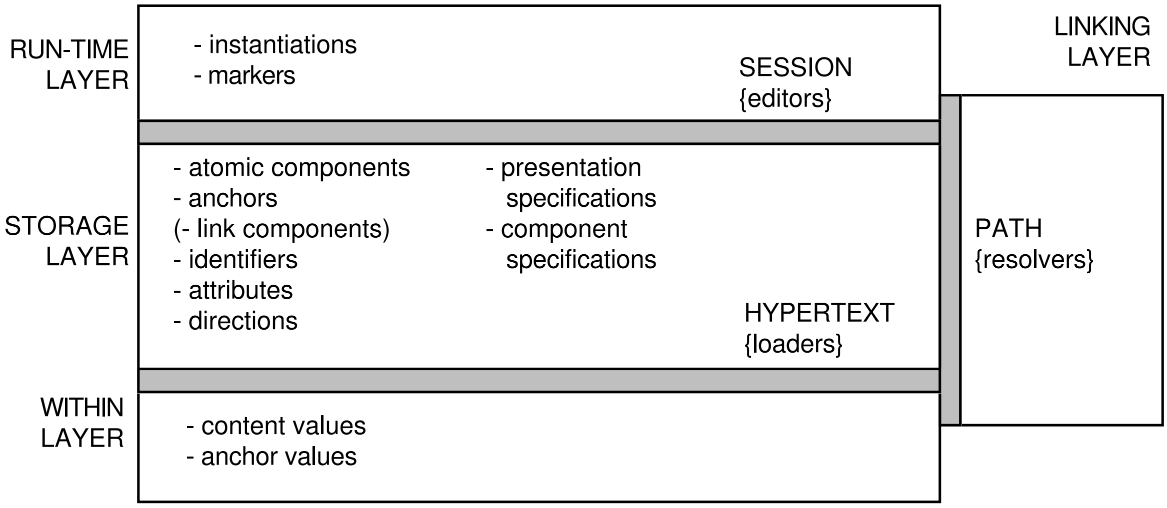

Figure 1: The extended Dexter model

Note that the introduction of a linking layer has an important side-effect: links become a less essential part of the hypermedia structure (in figure 1 the link components are surrounded by parentheses to represent this idea). Indeed, links are but one way to represent relations between components; resolvers are free to use different representations. This proves our claim that the introduction of paths ends "the tyranny of the link" [Halasz91]. Again, we will use the examples in section 4 to show the architecture can handle various link behaviour.

3.4. Wrap up

The insight that the Dexter architecture is unable to deal with the interoperability problem led to the introduction of loaders and editors communicating with external applications. A new linking layer containing the path concept (and the notion of resolvers) was added, in order to model an extensible link engine.

In the next section we will illustrate the possibilities of this open architecture, in particular the power of the extensible link engine, by means of a case-study involving a framework browser.

4. ILLUSTRATING EXAMPLES

To demonstrate the possibilities of the extended Dexter architecture we implemented a hypermedia system called *Zypher* based on the model proposed in section 3. Zypher is an extensible hypermedia system tailorable to specific application domains. Zypher is created in Smalltalk and takes the form of an object-oriented framework.

4.1. Frameworks, Design patterns and Framework browsers

We will use a software artefact called a framework browser to show that Zypher is able to incorporate various link resolving strategies and to co-operate with external applications. It is beyond the scope of this document to go into an extensive discussion of the software engineering concepts of frameworks, nor the techniques supporting the construction and documentation of frameworks. We refer the interested reader to [Johnson92], [Beck,Johnson94], [GammaEtAl93]. However, a brief explanation is needed to grasp the argumentation in the following sections.

3 Observant readers will notice that this violates the encapsulation enforced by the layered architecture. A Path is allowed to peek into other layers, because it is a meta-object in an open implementation [Rao91]. The technical argumentation is beyond the scope of this paper.

An *object-oriented framework* is a software engineering technique that supports software reuse in object-oriented programs. Classical object-oriented programming languages tend to focus on the reuse of code. Although this is generally considered a benefit, it is widely accepted that the reuse of design is more important. Object-oriented frameworks aim upon the latter. An object-oriented framework consists of classes, objects, methods, protocols that together form some kind of solution templates for a certain design space of problems. Filling in these templates results in an application that belongs to the design space. In the following, we will use the term *framework* to denote the concept of an object-oriented framework.

Since design is so important, an important stream of research in object-oriented frameworks investigates how to recognise the design in a large and complex object-oriented program. In these circles, people look at software design as an artistry rather than as an engineering activity. They start from the assumption that good designers have assembled a large collection of generally applicable solutions to solve certain kinds of problems. A *design pattern* is such a solution that is written down in a systematic way. Design patterns help a lot in communicating designs with other people and thus assist in describing the building blocks of an object-oriented framework.

A *framework browser* is a collection of software tools that helps programmers to inspect the design underneath the code that implements it. One possible element of a framework browser is a hypermedia system that links framework documentation (based on design patterns) to the framework code (i.e. the classes, objects and methods) that implements a certain aspect of the design space. But the development of a framework is an iterative group process with many tight loops: in such a dynamic environment, maintenance of static links between a framework and its documentation is not feasible. Links should be established on the fly, exploiting knowledge residing in the programming environment used to construct the framework. Therefore, a framework browser is an ideal case to demonstrate the power of an extensible link engine.

4.2. The Documentation Browser: HTML and Microsoft Word in Zypher

Our first step towards a framework browser is the construction of a set of document viewers that can be used to create and read framework documentation. We chose to adopt the popular HTML format [Berners-LeeEtAl94] as the basis for our documentation and integrated a home-cooked HTML-Viewer application in Zypher. This HTML-Viewer application was written in Smalltalk and it was fairly easy to incorporate it in Zypher.

But we were not satisfied with this solution because a monolithic hypermedia system is not the right approach for framework documentation. When one describes the design of a framework, one wants to apply all software tools available: word-processors, software design packages, CASE-tools, spreadsheets, ... So, framework documentation systems should allow to integrate viewers of the user's choice and not of the system's choice. Hence we integrated Microsoft Word as an additional viewer. We want to stress the fact that we can integrate other third-party applications in Zypher as well, so the Microsof Word application is just a representative for all other possible extensions.

To incorporate the third-party document viewer, we extended the Zypher-framework with a special purpose editor that communicated with Microsoft Word using DDE (Dynamic Document Exchange) under Windows. We used the macro facilities in Microsoft Word to accommodate for the necessary glue. Other authors ([Davis,Knight,Hall94], [Kacmar,Leggett91]) have shown that it is possible to integrate less co-operative applications as document viewers and the same techniques can be used to integrate them in Zypher.

The integration of a third-party application confronted us with "the Editing problem" [DavisEtAl92]: when documents are edited outside the hypermedia system the maintenance of the linkbase become a problem. Fortunately Zypher's extensible link engine allows us to choose an appropriate solution. We avoided the editing problem altogether by using embedded HTML links. Unlike traditional hypermedia links —which are stored separately from the documents— a HTML link is embedded in the document. The document contains references to other documents (a HTML anchor, not to be confused with the anchor-concept in the Dexter model) that are interpreted at run-time when such a reference is activated. A HTML anchor has a particular format of the form `<A href>marker</A>`, where 'href' represents the actual reference and 'marker' the visual representation of the anchor. HTML viewers are supposed to show the visual representation only; the parts between <> should be hidden. Note that the introduction of embedded links solves the editing problem but brings up the issue of maintenance of the overall information structure. In section 4.3 we will use the extensible link engine to implement an indirection technique that may help in maintaining references.

The result is depicted in figure 2: it shows the Microsoft Word application opened with a document containing HTML anchors. Note that we set the "show hidden text" option in the Microsoft Word-application so we are able to see the hidden parts (i.e. bold-dotted underline) of the HTML anchor; normally only the underlined part is visible. When a person reading the document places the cursor inside the visual part of the anchor and chooses the "Follow ZypherLink" in the File-menu, the reference in the anchor is interpreted and the corresponding document is opened using the appropriate viewer. In the example, when a user follows the link starting from the "(1)" anchor, the Microsoft Word document "ptt0100.doc" will be opened.

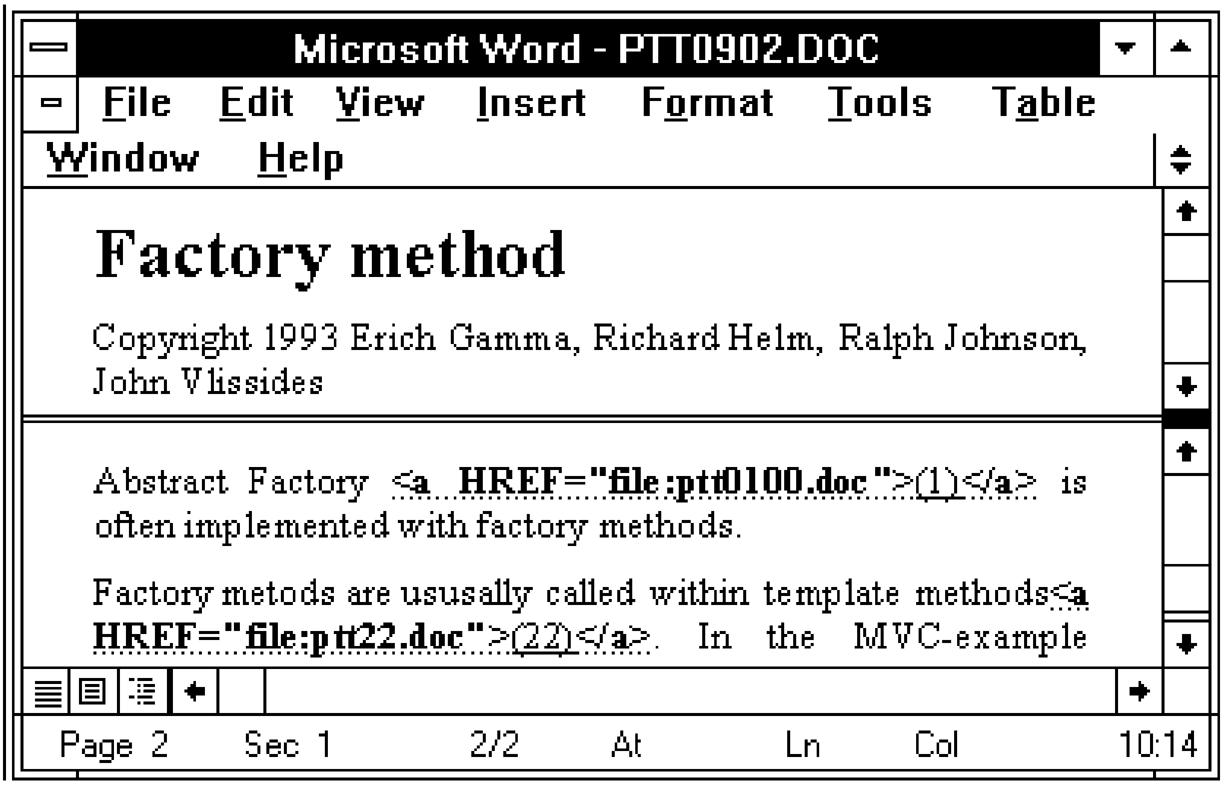

Figure 2: Microsoft Word document with HTML links

When implementing HTML resolvers for Zypher we were pleased to observe that we did not need the link concept in the original Dexter model, which is reasonable for an embedded link approach since all links are stored inside the document, not in a separate linkbase. This follows from the way Zypher handles embedded links, which is but a specific way for the overall handling of anchors. When an editor is opened on a document, the editor will pass all embedded link references to the path. The path will pass these references to the appropriate resolvers based on the first keyword in the reference. The resolver will create an anchor holding the reference string and return the anchors to the path that returns them to the editor. The editor will create markers as the run-time representations for these anchors. Later, when the editor activates such a marker, the path will pass the activation to the appropriate resolver (based on the linking tag associated with the anchor) which will interpret the reference string stored inside the anchor. No links are created in this process, which proves our claim that "paths end the tyranny of the link".

Note that the above scheme allows us to use the same resolver for both the HTML and Microsoft Word viewers (and in fact all other viewers that can hold text containing HTML anchors). All differences between the two document viewers are handled in the run-time layer. This is due to the layering-principle of the Dexter model.

4.3. Dynamic Structures: Named Patterns

In the previous section we have demonstrated that the Dexter extensions we propose allow us to cope with the interoperability problem and we saw a glimpse of the extensible link engine. This section will elaborate on the latter: the extensible link engine will be used to implement an indirection technique.

The link engine (= the path) in the previous experiment consisted of a set of named resolvers that interpret the classical HTML anchor types. There is one resolver for each possible anchor type: file, http, ftp, gopher, ... The path can easily pass each HTML anchor to the appropriate resolver since each HTML anchor is preceded by a keyword that represents its type. For all following experiments we will augment the set of acceptable keywords to extend the link engine. Although this says a lot about the extensible nature of HTML links, we want to point out that this does not diminish the value of the Zypher system. The open format of HTML links simply allows us to integrate new resolvers easily because we can use the existing keyword recognising machinery. We can apply other approaches to attach types to anchors inside documents (i.e. an extensible menu in each viewer application containing the names of all resolvers) but we chose not to in order to avoid "the Editing problem" mentioned above.

It is beyond the scope of this paper to say more about how Zypher is extended. We refer the interested reader to [Grønbaek,Malhotra94] and [Wiil,Legett92] for a description of techniques that enable system extensions without accessing the source-code of the overall implementation. Since Zypher is implemented in Smalltalk similar techniques can be applied. We point out that Zypher is run-time extensible and that adding new resolvers does not affect the rest of the implementation.

Being able to navigate dynamic information structures inside a hypermedia system is our prime motivation for an extensible link engine. If we re-examine the previous experiment we see that the information structure is very rigid in nature. In figure 2 the link that starts at (1) is actually a reference to some file in the file system that should be opened with the Microsoft Word viewer. But the author of the link intended this to be a reference to a design pattern with the name "Abstract Factory". The limitations of the underlying file system forced him to choose some bizarre naming convention which hinders the maintenance of the documentation. Since framework documentation evolves rapidly, one can not permit the limitations of the underlying file system to obstruct the maintenance of the links.

So we plugged in an extra "pattern" resolver that is able to resolve design pattern references by name. For the example in figure 2 the anchor is modified to the form `<a HREF = "pattern: abstract factory">`. The resolver inspects a table maintained in a simple text-file to map pattern names on the actual file references. Moreover, the resolver inspects the state of the Zypher system to choose the appropriate document viewer. It chooses the Microsoft Word editor when available, otherwise it uses the ordinary HTML Viewer[4].

The pattern resolver brings up the issue of composite resolvers. Just like the notion of composite components (a component that is composed of other components) in the original Dexter model, we can imagine a composite resolver that is composed of other resolvers. The pattern resolver is in fact a composite resolver that glues the effects of a name mapping resolver with those of a file reference resolver. The potential of composite resolvers is enormous as we can create custom resolvers by composing off-the shelf resolvers with special abilities (i.e. query databases, inspect information retrieval structures, ...). Such composite resolvers would make it easier to tailor a hypermedia system to a specific application domain, as one can reuse existing resolvers.

4.4. More Dynamic Structures: Querying the Programming Environment

Currently we have demonstrated that the extended Dexter model we propose allows us to cope with the interoperability problem and handle an extensible link engine. It is now time to show that the real power of such an extensible system lies in the fact that we are able to exploit knowledge inside external information repositories to construct hypermedia structures. This allows a hypermedia system to browse complex and dynamic information structures maintained outside the hypermedia system, which is important when one wants to tailor a hypermedia system to a specific application domain.

To illustrate this property we extended our Zypher system from the previous sections so that we are able to link HTML documents with the Smalltalk code browser. The Smalltalk code browser is the primary medium to inspect and change Smalltalk programs. Unlike traditional programming environments that treat programs like large sequential texts, the Smalltalk browser is very "hypertexty". We refer to [Goldberg84] for more information on how the code browser actually works.

For the experiment, we adapted the format of the reference inside HTML-anchors again. The keyword "browse:" is followed by references like "system" (to open the global system browser), "class X" (to open a class browser on the class X), "class X method Y" (to open a method browser on the method Y in class X), "sendersOf X" (to browse all methods that send the message X), "implementorsOf X" (to browse all methods that implement the message X). Then we extended the Zypher implementation with

[4] Currently, we use Microsoft Word as a document viewer on Windows-machines. All other platforms use the home-cooked HTML-Viewer to read framework documentation.

an extra resolver that is able to interpret such references and opens the requested Smalltalk browser. The result is depicted in figure 3.

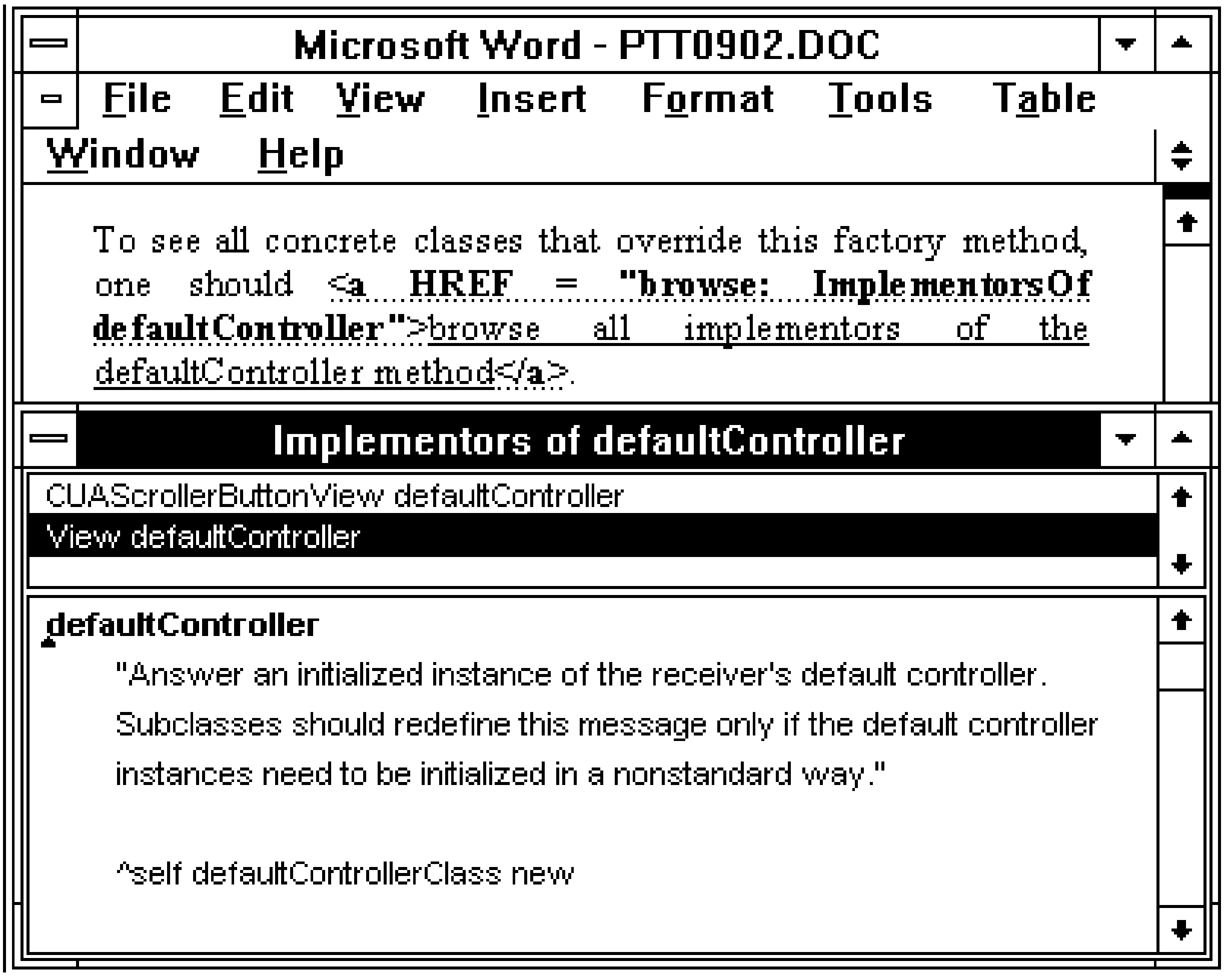

figure 3: The framework browser in action

The window in the back of figure 3 is a Microsoft Word document containing the "Object Factory" design pattern [GammaEtAl93]. The pattern contains an example on the application of object factories in the MVC-Framework [Goldberg84]. The document-window is focused on the description of how the link between the view- and controller-object is established. Again, we turned on the "show hidden text" option in the Microsoft Word viewer so we are able to see the hidden text (i.e. dotted underline) containing the HTML anchor `<a href = "browse: implementorsOf defaultController"> ... </a>`. Following the link starting in this anchor opens a Smalltalk method list browser (the front-most window). The top pane of this window contains a list of all `defaultController` methods in the Smalltalk system, the bottom one contains the source code of the selected method.

Note that the contents of the method list browser depends entirely on the Smalltalk system. At the time the link is established, the Smalltalk system will collect all methods that satisfy the query and present them using the browser. If a programmer chooses to override the `defaultController` method in some other class, the next time the link is followed the method list browser will contain the additional method. This proves that —due to the extensible link engine— we are able to exploit knowledge inside external information repositories to maintain hypermedia structures.

Nevertheless, explicit class- and/or method names are still needed to establish a link between the documentation and the framework. This resembles the situation in 4.2 where explicit file names were used to refer to higher level concepts. This hinders the maintenance of the documentation. Currently, we are investigating how to extend the resolver with extra querying facilities that provide more support for framework browsers. We want to use the implicit contracts defined in the design patterns so that the explicit references to class- and method-names can be omitted from the documentation.

The experiment teaches us that the Smalltalk code browser is just another third-party viewer. It should be possible to create and follow links from the Smalltalk code browser to the documentation and back again. Such a set-up would constitute a real framework browser.

Of course it is possible to integrate the Smalltalk code browser into the Zypher hypermedia system. In fact it is fairly easy, since the code browser can be tailored using the programming environment itself. But this is not what we plan to do since this would restrict the possible views on the framework. As is, the Smalltalk code browser provides a limited, class-oriented view on the framework structure. Since framework design is usually spread over several classes, a code browser should be able to inspect different classes at the same time in the same viewer. Thus the layout of a code browser should change depending on the particular viewpoint on that framework. Currently, we investigate how we can use Zypher's dynamic link engine to model structures inside object-oriented frameworks. We plan to implement code browsers that build on this knowledge to display the important aspects and hide the others.

4.5. Wrap up

Section 4.1 studied object-oriented frameworks, framework browsers and design patterns. Those terms were needed to understand the case-study in the following sections. Section 4.2 described how we incorporated documentation viewers in the Zypher system. Microsoft Word was chosen as a representative for all third-party applications that can be used to manipulate framework documentation. The experiment proved that Zypher is able to handle the interoperability problem. Embedded (HTML style) links are used to avoid the problems with documents that can be edited outside the hypermedia system. Section 4.3 contained a first example of the extensible nature of the link engine. A composite resolver was used to map pattern name on actual documents. Composite resolvers enable the construction of new resolvers from off the shelf resolvers. Section 4.4 showed that Zypher's extensible link engine is actually capable of exploiting information residing in external information sources. It uses the Smalltalk code browser as another third-party application to view selected elements of the framework.

5. RELATED WORK

5.1. Extensible link engines

There are lots of Open Hypermedia Systems around, most of them have been named in other sections. However, few of them mention an extensible link engine as an explicit feature. Nevertheless, systems like DHM ([Grønbaek,Trigg94], [Grønbaek,Malhotra94]), Multicard [Rizk,Sauter92], Proxhy ([Kacmar,Leggett91]), SP3 ([Legget,Schnase94]) are based on a software architecture that could be extended to provide for special purpose links. Also, all hypermedia systems supported by a scripting language (such as MacWeb [Nanard,Nanard93] and KMS [Ackscyn,McCracken,Yoder88]) are able to tailor their link behaviour, but this is a side-effect rather than an explicit design feature.

WWW-browsers can extend their link engines as well. The protocol supports communication with external applications, both on the server and on the client side. To extend the link engine, one will augment the set of acceptable keywords like we described in 4.3. The server must then be adapted to pass hypertext references starting with such a new keyword to a third party application that handles it. Some technical expertise is required in this process though. WWW-browsers have the fundamental problem that they are used for navigation only; modifying the structure should be done outside the system.

Microcosm [Davis,Knight,Hall94] is —to our knowledge— the only system that supports a truly extensible link engine. Viewer applications send tagged messages to Microcosm. Those messages are passed through a filter-mechanism that interprets the messages and returns commands for the viewer applications. However, we do not approve the message-format, because it violates the encapsulation principle of the Dexter model (run-time layer aspects are completely mixed up with storage layer aspects).

5.2. Dexter extensions

There are other people working on extensions of the Dexter model. DHM ([Grønbaek,Trigg94]) started from the Dexter specification, but stopped being Dexter compliant when investigating CSCW [Grønbaek,Hem,Madsen94] and third-party application integration [Grønbaek,Malhotra94]. Work was done on the composite-problem [Grønbaek94] as well. The Dexter model has been tackled from a multi-media perspective by [Hardman,Bulterman,VanRossum94] and [Garzotto,Mainetti,Paoline94]. The former adds timing information to links in order to synchronise multi-media presentations and uses special purpose composites as a multi-media authoring aid. The latter adopt composites (collections) to model new kinds of structures and navigation and shows that there are other possibilities besides links as the basis for navigation.

Especially DHM has much in common with our work. It is an object-oriented implementation for the Dexter specification using advanced software engineering techniques like meta-object protocols [Rao91]. The system allows for third party application integration, partially based upon the anchor and presentation specifications from the original Dexter design. The implementors saw that the Dexter model was underspecified and tried to correct this to suit their needs. However, DHM concentrates on an extended composite-concept to model dynamic structures, while we focus on the link engine. The two approaches are not necessarily incompatible.

5.3. Hypermedia and Framework browsers

Throughout the years, there have been several experiments to apply hypermedia technology in the field of Software Engineering: we will mention but a few. Chimera [Anderson,Taylor,Whitehead94] is —to our knowledge— the only system that demonstrates how third-party applications can be used to enrich software development environments. Hyperpro [Østerbye,Nørmark94] is used to link documentation to source-code; a rule-system controls the way the source-code is presented to the user. The combination of the rule-system and the link engine is an example of dynamic links as defined in section 2.2. ChyPro [Amsellem95] applies hypermedia techniques to gain flexibility in a Smalltalk programming environment, somewhat similar to that of Hyperpro. The ET++ Hypertext system [Pree94] is the only hypertext system that is explicitly related with the notion of object-oriented frameworks and design patterns. The hypertext system was constructed to test the extensibility of the ET++ framework, not to supply electronic documentation for an object-oriented framework.

The literate Smalltalk Programming Environment described in [Østerbye95] comes close to what we call a framework browser. It consists of a simple hypertext model that supports text nodes (with embedded anchors) as well as nodes representing Smalltalk classes and methods. The text-nodes are supposed to contain documentation about the Smalltalk program at hand and contain embedded links that bring the user to the actual Smalltalk code. The links that structure the Smalltalk nodes incorporate knowledge from the Smalltalk environment and dynamic (query) links are used to find all senders of .

6. EXTENSIBLE LINK ENGINES REVISITED

We claim that an extensible link engine is important because it permits a hypermedia system to choose the linking model that is appropriate for the information structure it represents. This is inevitable if one wants to tailor a hypermedia system to a specific application domain.

Although we did not show that we can deal with exotic information structures like the set-based model of Parunak [Parunak91] or the spatial layouts of Aquanet [Marshall,Shipman93], we did show that paths absorb links (4.2) and model an extensible link engine that is able to access external information repositories (4.3 & 4.4).

The question rises whether this is still hypermedia. After all, if one uses external information repositories to resolve links, one comes pretty close to what is commonly regarded as an information retrieval system.
The short answer is that this is not a relevant question. Hypermedia and information retrieval have always been overlapping research domains and this will remain so in the future. If one calls Microcosm [Davis,Knight,Hall94] and QRL [CharoentkitkarnEtAl93] hypermedia systems, then so is Zypher.
The long answer is that it is precisely the extensibility and interoperability that makes Zypher a hypermedia system, not an ordinary information retrieval system. The extensible link engine is able to handle the static link/node model as well as it allows for search-and-query types of links. Static links can not be omitted from a real hypermedia system because they allow humans to annotate information, to form trails, to organise data, in short to apply their human intellect to relate chunks of data to form information. Machines will never be able to be as creative as humans, so we must leave the ability to create simple static links. But using static links to maintain dynamic structures is not possible. So static and dynamic 'links' should go side by side. The interoperability aspect is essential because it offers the extra degree of freedom that separates hypermedia from other information manipulation systems. The way the information is presented should be a users decision, not a systems decision.

7. CONCLUSION

With others in the hypermedia community, we believe that hypermedia technology is a potential benefit for all computer applications that deal with information. The benefit offered by hypermedia is the extra degree of freedom that makes it possible to personalise information processing [Ashman94]. A user is free to explore that fragment of the information he chooses with whatever tool he finds appropriate. However, this benefit comes at a cost: hypermedia technology implies more complexity. From a user's perspective, the freedom of choice imposes a mental overhead, giving rise to the "lost in hyper space" phenomenon. From a system implementor's perspective this complexity requires extra programming efforts that are far too expensive. The work described here aims to solve the last problem.

A tool kit approach can break the system implementor's complexity barrier. Application developers may use the functionality supplied by the tool kit whenever appropriate. The real issue for Open Hypermedia Systems is then how to ensure that the tool kit functionality is appropriate for a wide range of application domains. Integrating third-party applications is an important aspect of the problem, but the incorporation of external information structures is at least as important. A case study in Software Engineering (framework browsers) supports our claim that an extensible hypermedia system equipped with an extensible link engine is the right tool to build hypermedia systems that can be tailored to any application domain.

8. ACKNOWLEDGEMENTS

Much of this work is influenced by the Workshop on Open Hypermedia systems [Wiil,Østerbye94]. Kasper Østerbye and Uffe Kock Wiil (the organizers of the workshop) did a wonderful job in reviewing an earlier draft of this document.

All of my colleagues and all of the proof-readers (Wolfgang De Meuter, Carine Lucas, Tom Mens) are to be acknowledged, but Patrick Steyaert and Koen De Hondt deserve special attention. I am grateful to Professor Theo D'Hondt for setting up an inspiring environment in which this work has found its roots. The experiments concerning HTML are based on work Mark Plas.

9. REFERENCES

[Ackscyn,McCracken,Yoder88] Ackscyn, R. M. / McCracken D. L. / Yoder, E. A. "KMS: A distributed hypermedia system for managing knowledge in organizations"; Communications of the ACM - Vol. 31(7) - July '88. Also in Hypertext '87 Proceedings, ACM Press, 87.

[Amsellem95] Amsellem, M. "ChyPro: A Hypermedia Programming Environment for Smalltalk-80"; ECOOP'95 Proceedings, Lecture Notes in Computer Science nr. 952, Springer-Verlag, 1995.

[Anderson,Taylor,Whitehead94] Anderson, K. M. / Taylor, R. N. / Whitehead, E. J. "Chimera: Hypertext for Heterogeneous Software Environments"; ECHT'94 Proceedings, ACM Press, 94.

[Ashman94] Ashman, H. "Opinion: What is Hypermedia"; ACM SIGLINK Newsletter - Vol. 3(2) - September '94.

[Beck,Johnson94] Beck, K. / Johnson, R. "Patterns Generate Architecture"; ECOOP'94 Proceedings, Lecture Notes in Computer Science nr. 821, Springer-Verlag, 1994. Check http://st-www.cs.uiuc.edu /users /patterns /patterns.html for anonymous ftp.

[Berners-LeeEtAl94] Berners-Lee / Cailliau, R. / Luotonen, A. / Nielsen, H. F. / Secret, A. "The World-Wide Web"; Communications of the ACM - Vol. 37(8) - August '94.

[Bieber,Kacmar95] Bieber, M. / Kacmar, C. "Designing Hypertext Support for Computational Applications"; Communications of the ACM - Vol. 38(8) - August '95.

[CharoentkitkarnEtAl93] Charoenkitkarn, N. / Tam, J. / Chignell, M. H. / Golovchinsky, G. "Browsing Through Querying: Designing for Electronic Books"; Hypertext'93 Proceedings, ACM Press, 93.

[Davis,Knight,Hall94] Davis, H. / Knight, S. / Hall, W. "Light Hypermedia Link Services: A Study of Third Party Application Integration"; ECHT'94 Proceedings, ACM Press, 94.

[DavisEtAl92] Davis, H. / Hall, W. / Heath, I. / Hill, G. "Towards An Integrated Information Environment with Open Hypermedia Systems"; ECHT'92 Proceedings, ACM Press, 92.

[DuvalEtAl95] Duval E. / Olivie H. / O'Hanlon P. / David G. Jameson "HOME: an Environment for Hypermedia Objects" Journal of Universal Computer Science - Vol 1(5) - May 1995. Available from http://www.iicm.tu-graz.ac.at/Cjucs_root. Also "HOME: Hypermedia Object Management Environment"; Demonstration at ECHT'94.

[GammaEtAl93] Gamma, E. / Helm R. / Johnson R. / Vlissides, J. "Design Patterns: Abstraction and Reuse in Object-Oriented Designs"; ECOOP'93 Proceedings, Lecture Notes in Computer Science nr. 707, Springer-Verlag, 1993. The same people have written the book "Design Patterns"; Addisson-Wesley, 1995.

[Garzotto,Mainetti,Paoline94] Garzotto, F. / Mainetti, L. / Paoline, P. "Adding multimedia Collections to the Dexter Model"; ECHT'94 Proceedings, ACM Press, 94.

[Goldberg84] Goldberg, A. "Smalltalk-80: The Interactive Programming environment"; Addisson-Wesley, Reading MA, 1984.

[Grønbaek,Hem,Madsen94] Grønbaek, K. / Hem, J. A. / Madsen, O. L. "Designing Dexter-based Cooperative Hypermedia Systems"; Communications of the ACM - Vol. 37(2) - February '94. Also in Hypertext'93 Proceedings, ACM Press, 93.

[Grønbaek,Malhotra94] Grønbaek, K. / Malhotra, J. "Building Tailorable Hypermedia Systems: the embedded-interpreter approach"; OOPSLA '94 Proceedings; ACM Press, 1994.

[Grønbaek,Trigg94] Grønbaek, K. / Trigg, R.H. "Design issues for a Dexter-Based Hypermedia System"; Communications of the ACM - Vol. 37(2) - February '94. Also in ECHT'92 Proceedings, ACM Press, 92.

[Grønbaek94] Grønbaek, K. "Composites in a Dexter-Based Hypermedia Framework"; ECHT'94 Proceedings, ACM Press, 94.

[Halasz,Schwartz90] Halasz, F. / Schwartz, M. "The Dexter Hypertext Reference Model"; Proceedings of the 1990 NIST Hypertext Standardisation Workshop (January 16-18, Gaithersburg, MD). Republished in Communications of the ACM - Vol. 37(2) - February '94.

[Halasz91] Halasz, F. "Seven issues revisited"; Slides from the ACM Hypertext '91 Conference Keynote speech (December 15-18, San Antonio, Texas).

[Hardman,Bulterman,VanRossum94] Hardman, L. /Bulterman, D. C. A. / Van Rossum, G. "The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter model"; Communications of the ACM - Vol. 37(2) - February '94.

[Johnson92] Johnson, R. "Documenting Frameworks Using Patterns"; OOPSLA'92 Proceedings, ACM Press, 1992. Check http://st-www.cs.uiuc.edu /users /patterns /patterns.html for anonymous ftp.

[Kacmar,Leggett91] Kacmar, C. J. / Legget, J. J. "PROXHY: A Process-Oriented Extensible Hypertext Architecture"; ACM Transactions on Information Systems - Vol. 9(4) - October '91.

[Legget,Schnase94] Legget, J. J. / Schnase, J. L. "Viewing Dexter with Open Eyes"; Communications of the ACM - Vol. 37(2) - February '94.

[Marshall,Shipman93] Marshall, C. C. / Shipman, F. M. "Searching for the Missing Link: Discovering Implicit Structure in Spatial Hypertext"; Hypertext'93 Proceedings, ACM Press, 93.

[Nanard,Nanard93] Nanard, J. / Nanard, M. "Should Anchors Be Typed Too ? An experiment with MacWeb"; Hypertext'93 Proceedings, ACM Press, 93.

[Østerbye,Nørmark94] Østerbye, K. / Nørmark, K. "An Interaction Engine for Rich Hypertexts"; ECHT'94 Proceedings, ACM Press, 94.

[Østerbye95] Østerbye, K. "Literate Smalltalk Programming Using Hypertext"; IEEE Transactions on Software Engineering - Vol. 21(2) - February '95.

[Parunak91] Parunak, H.V.D. "Don't Link Me In: Set based Hypermedia for Taxonomic Reasoning"; Hypertext'91 Proceedings, ACM Press, 91.

[Pree94] Pree, W. "Design Patterns for Object-Oriented Software Development"; Addisson-Wesley 1994.

[Rao91] Rao, R. "Implementational Reflection in Silica"; ECOOP'91 Proceedings, Lecture Notes in Computer Science nr. 512, Springer-Verlag, 1991.

[Rizk,Sauter92] Rizk, A. / Sauter, L. "Multicard: An open hypermedia System"; ECHT'92 Proceedings, ACM Press, 92.

[Stotts91] Stotts, P. D. "Dynamic Adaption of Hypertext Structure"; Hypertext'91 Proceedings, ACM Press, 91.

[Wiil,Legett92] Wiil, U. K. / Legett, J. J. "Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems"; ECHT'92 Proceedings, ACM Press, 92.

[Wiil,Østerbye94] Wiil, U. K. / Østerbye, K (editors) "Proceedings of the ECHT'94 Workshop on Open Hypermedia Systems"; Technical report R-94-2038 / Institute for Electronic Systems Department of Mathematics and Computer Science - Fredrik bajers Vej 7 - DK 9220 Aalborg - Denmark.