CHAPTER *1*

Open Hypermedia

Not so many years ago, it would have taken quite a long time to explain to people the basic notions of hypertext (or hypermedia — both names are used interchangeably throughout this text). Oh yes, system administrators would have used technical documentation in hypertext form and some Macintosh users might even had experiences with HyperCard stackware. Still, most people would have shown big question marks hearing definitions like "hypermedia is a style of building systems for information representation and management around a network of multi-media nodes connected together by typed links" [Halasz'87] and certainly would not see the tremendous possibilities of what has been called the navigational paradigm to computing.

Today, this is different. With the coming of the world-wide web, almost all computer users are well aware of basic hypermedia concepts like links, anchors, nodes, non-linearity and navigation. The establishment of cyber cafes takes away the obstacle of not owning a computer: people can visit specially designed places to surf the web. In the United States, and more and more in Europe, the URL's are an integral part of advertising and hypermedia is even becoming visible in the streets now.

If hypermedia has finally hit the mass market, then why is it still appropriate to write a Ph.D. dissertation on such a topic ? Well, as you will notice reading this text, hypertext and hypermedia research has a long standing history of fruitful research and the computer science community can —and will— pluck more of those inspiring ideas in the time to come. One of this ideas is open hypermedia, and that is the topic discussed in this chapter.

Hypertext & Hypermedia

With the explosive growth of the world-wide web, people are generally acquainted with the basic hypermedia concepts. Still, there is more to hypermedia than there is to the world-wide web and this section introduces many of the important ideas living in the hypermedia community.

There are several ways to give an overview of those ideas. One could try for an analytical approach by partitioning the space of hypermedia systems along a number of dimensions. This approach has been taken in [Halasz'87], [Halasz'91] and also in [Akscyn,McCracken,Yoder'87], [Akscyn'91].

The problem with an analytical approach is that it is hard to illustrate why a certain dimension is 'good' way to partition the space. To avoid this problem, we chose to provide a definition and basic vocabulary of hypermedia, followed by brief historical overview that introduces the reader to the major milestones in hypermedia research. As a rough guide we classified the milestones in generation of hypermedia systems. As with all brief overviews, many contributions of many interesting projects could not be included, so be aware that this is by no means the ultimate overview of hypermedia research.

What is Hypermedia ?

Hypertext, Non-linearity and the Navigational Paradigm

The simplest way to define hypertext is to contrast it with the more traditional approaches to text. Traditional information sources like books are based on the idea of a linear processing of information: readers are supposed to read from the first page to the last page. Through the history of paper publication, people have found ways to circumvent this linear processing of ideas (i.e. a phone book, a dictionary, a table of contents, an index, …) and the invention of desktop computers opened a new range of promising solutions.

Hypertext is the name of the research domain that investigates the possibilities of manipulating non-linear text: the idea is that computers can be used to help users create, maintain and navigate between chunks of text to form structures and shape information. Navigation is the basic operation that brings information explorers from one chunk to another, which explains why hypertext is sometimes called the navigational paradigm.

Note that, although hypertext contrasts with traditional text approaches, a lot of the traditional vocabulary of sequential text remains appropriate. The hypertext community adopts terms like document, reading, reader, authoring, author and viewer instead of the more computer oriented ones like file, using, user, implementing, implementor and application.

Hypermedia = Hypertext + Multi-media

The idea of hypertext is quite old, but with the advent of powerful workstation technology, it became feasible to extend the notion of hypertext beyond mere text-processing. Instead of manipulating chunks of text, one could also shape information structures containing pictures, video, sound, …. . Hypermedia —a contraction of the words Hypertext and Multi-media— is a name invented to stress this change of emphasis. However, like most material in the field, we use both terms interchangeably.

Node, Link and Anchor

Many hypermedia projects have attempted to build models to support this navigational paradigm. Each project invented its own terminology resulting in an enormous amount of words referring to similar concepts. Still, the hypermedia community has managed to reach consensus on some basic vocabulary. As the first hypermedia projects started from a data model based on graphs, the basic vocabulary inherits quite a lot from graph theory.

The basic unit of information in a hypermedia system is called a node, which may contain arbitrary information like text, graphics, video, sound, … . Some hypermedia systems allow a special kind of node, called a composite node, which can contain other nodes.

A hypermedia system organises nodes into navigational structures. A link is the name for an explicit representation of a navigation relationship that exists between nodes. The navigation effect —essential to all hypermedia systems— is achieved by traversing the links. Links can be binary or n-ary; the latter allows an arbitrary number of endpoints for a link, the former allows exact two. Most hypermedia systems enforce directed links, which can be traversed in one direction only. The traversal direction of the link determines a source (the starting point of the navigation operation) and a destination (the arrival point). Many hypermedia systems allow typed links, which have some kind of label attached to them so that the reader may infer the purpose of the link without actually traversing it.

Links can be connected to whole nodes, but in most occasions links have special purpose connection points called anchors. An anchor is thus an explicit representation of a referencable substructure of a node serving as an endpoint of a link.

Hypertext & Hypermedia Definition

In this dissertation, we adopt the definition of the influential "Seven Issues: revisited" keynote address [Halasz'91].

"Hypermedia is a style of building systems for the creation,manipulation,presentation and representation of information in which:
- *the information is stored in a collection of multi-media nodes*
- *the nodes are explicitly or implicitly organised into one or more structures (commonly, a network of nodes connected by links)*
- *users can access information by navigating over or through the available information structures."*

There are two important remarks to make on this definition. First of all, it is a revised version of the definition that appeared in [Halasz'87]. The original version reads as follows: "Hypermedia is a style of building systems for information representation and management around a network of multi-media nodes connected together by typed links". In the revised version, Halasz explicitly recognises that the means to construct navigational structures (i.e. nodes and links) are less important than the idea of navigation itself. This change of emphasis was driven by promising new approaches for structuring navigation relationships (i.e. spatial hypertexts).

The second remark is that one cannot underestimate the importance of the word "style" in this definition. To quote Jacob Nielsen [Nielsen'90] - p.02 "*Many non-hypertext computer techniques may at least match various aspects of the definition of hypertext, but true hypertext should also make users feel that they can move freely through the information according to their own needs. […] When asked whether I would view a* *certain system as hypertext, I would not rely so much on its specific features, commands, or data structures, but more on its user interface 'look and feel'.*"

On this matter of style the last word is not said. Perhaps the closest specification of the essence of hypertext style is the one from Helen Ashman [Ashman'94]: "*The common thread that runs through most definitions is the idea that hypermedia permits users to customise the presentation of the information in some way. At least some part of the information presentation is in the user's control. In the simplest cases, only the order of traversal is decided by the user. But the more complex and powerful hypermedia systems let the user add to the hypermedia link structures as well. In every case, the user creates a personalised environment tailored to meet their needs. It is this personalisationand customisability of the information presentation that characterises hypermedia. This incorporates the idea of non linearity, since non linearity gives control of the order of traversal to the user*".

The style aspect in the definition is a subjective criterion and makes it very hard to judge whether a system is hypertext and hypermedia. Nevertheless, or maybe just as a consequence, the lack of a precise definition has not tempered the hypermedia community to provide numerous fruitful ideas and we discuss the most important ones below.

A Guided Tour of Hypermedia

The earlier periods of this guided tour are primarily based on the book "Hypertext & Hypermedia" by Jacob Nielsen [Nielsen'90]. Other inspiration was drawn from the annual conferences on Hypertext: the first two organised in the US [HT'87], [HT'89] and from then on alternating between Europe and the US [ECHT'90], [HT'91], [ECHT'92], [HT'93], [ECHT'94], [HT'96]. The keynote address by Randy Trigg [Trigg'96] served as a backbone for the classification in generations.

a) The Pioneers — Memex, Augment and Xanadu

Memex — Vanevar Bush (1945)

Most people agree that Vanevar Bush is to be considered the founding father of hypermedia research. His paper "As we may think" [Bush'45], described an imaginary "Memex" system based on microfilm technology (!) to support scientists in their task of tracing the enormous amount of information that was published in their discipline. The idea was that people could use a machine to link together fragments of different documents and pass this annotated material to colleagues: the very same idea that started of the world-wide web project [Berners-LeeEtAl'94].

Augment/NLS — Douglas Engelbart (1962-1976)

While the Memex system coined the idea of using a machine to maintain associative links between document fragments, it was the Augment project that —in a time where computers were used mainly as number crunchers and instructed through punch cards— laid the basis for computer technology as tools for realising a "Conceptual Framework for Augmenting Human Intellect" [Engelbart'95]. This project introduced a second major theme of the future hypermedia research, that of computer supported co-operative work (CSCW). Besides that, the Augment project invented nearly half of the concepts of today's desktop environments, such as e-mail, teleconferencing, windows, the mouse pointing device, etc.

The hypermedia research community still acknowledges the great influence of the vision of Douglas Engelbart and has established a yearly prize in honour of his person.

Xanadu — Theodor Nelson (1965-…)

It was the Xanadu project that actually coined the words hypertext and hypermedia in a vision that all information would be published in a universal information repository, accessible via computers ("literary machines"). Although the Xanadu Operating

Company (see http://www.mpx.com.au/) is entirely devoted to the implementation of that vision, it was the world-wide web project [Berners-LeeEtAl'94] that came nearest to a such a universal information space. Nevertheless, the Xanadu project will always be credited as the first project that performed constructive thinking on problems like universal referencing schemes [Nelson'87], copyright issues, versioning, etc.

The three pioneering hypermedia systems share the idea of using machines to link together fragments of information. Yet, they came to this idea from a completely different angle. Memex envisioned a system supporting scientists in organising information, so the emphasis was on authoring associative links and on the annotation of material. Augment/NLS was a prototype of an office information system, where co-operating people work on shared material, hence the need for fine grained concurrency control mechanisms, integration of tools and user-interface. Xanadu aimed for very large and distributed information systems, so there the emphasis was more on coarse grained concurrency control algorithms, replication of information and referencing schemes.

b) The Monolithic Hypermedia Systems — NoteCards and Intermedia

After these three pioneering projects, lots of activity went on in hypermedia community. We refer the interested reader to [Conklin'87] for an overview and a comparative study of those earlier systems (with names like Boxer, CREF, Emacs INFOR, IBIS, Intermedia, KMS, Neptune, NoteCards, PlaneText, Document Examiner, SYNVIEW, Textnet, Hyperties, WE — and the list is not yet complete) and concentrate on two of them: NoteCards and Intermedia. Our choice was guided by the significant impact of both systems on all later hypermedia research. Based on the classification of [Trigg'96], we see them as representative for the generation of monolithic hypermedia systems.

NoteCards

NoteCards [Halasz'87] was developed as a research prototype at Xerox Parc within the InterLisp environment. Programmers were able to extend the system by means of an API of over a 100 routines so that the NoteCards system could be tailored to their needs. The NoteCards prototype has been used outside Xerox Parc in numerous settings and as such the system had quite a lot of impact on the hypermedia community.

The design of the system was based on a 'notecard': an electronic generalisation of the 3x5 paper notecard that could contain an arbitrary amount of some editable information (text, graphics, …). Cards could be linked together by means of typed directional links; users could follow links by clicking on the source of a link to pop up the target card. The 'browser' allowed users to edit a network of interlinked notecards and the 'filebox' enabled the collection of notecards into composites.

Hypermedia Research Agendas

The main reason why the NoteCards system could not be omitted from this overview is the fact that it was used twice as a foil against which to explore some of the major limitations of two generations of hypermedia systems and as such caused the definition of two agendas still driving the research in the hypermedia community. The first (see [Halasz'87]) put forward a list containing the items: (1) search and query, (2) composite nodes, (3) virtual structures, (4) computational engines, (5) versioning, (6) collaborative work and (7) tailorability. This agenda was revised in 1991 (see [Halasz'91]) in the list: (1) ending the tyranny of the link, (2) open systems, (3) support for collaborative work, (4) user interfaces for large information spaces, (5) very large hypertexts, (6) tailorability and extensibility, (7) computation in (over) hypermedia networks, (8) defining the hypermedia markets, (9) standards, (10) publishing hypertexts. Figure 1 shows how both research agendas map onto each other. It would take us too long to study each of the items in detail, but some of them are covered in the section open hypermedia (p.22).

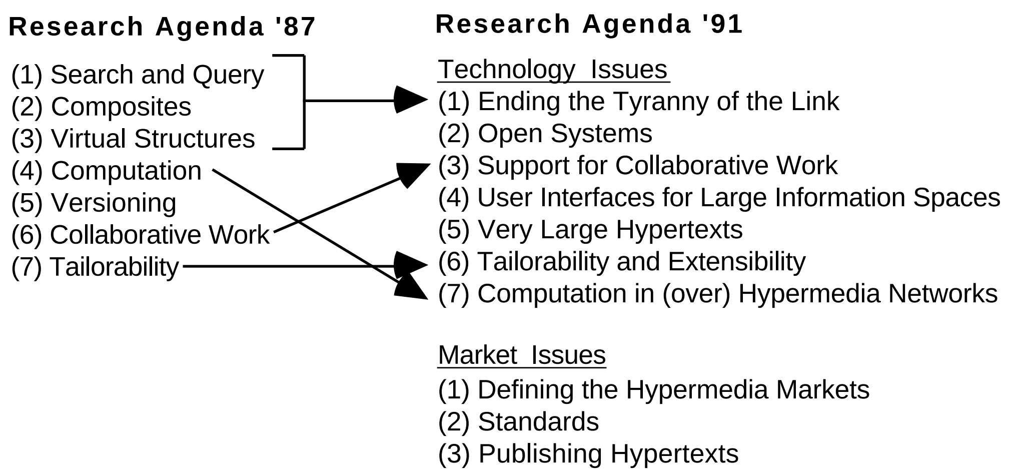
Research Agenda '87
(1) Search and Query
(2) Composites
(3) Virtual Structures
(4) Computation
(5) Versioning
(6) Collaborative Work
(7) Tailorability
Research Agenda '91
Technology Issues
(1) Ending the Tyranny of the Link
(2) Open Systems
(3) Support for Collaborative Work
(4) User Interfaces for Large Information Spaces
(5) Very Large Hypertexts
(6) Tailorability and Extensibility
(7) Computation in (over) Hypermedia Networks
Market Issues
(1) Defining the Hypermedia Markets
(2) Standards
(3) Publishing Hypertexts

Figure 1: Hypermedia Research Agendas

Intermedia

Intermedia ([Yankelovich,Meyrowitz,VanDam'85], [Meyrowitz'86], [Catlin,Bush,Yankelovich'89], [HaanEtAl'92]) is another hypermedia system that has influenced —directly or indirectly— all subsequent hypermedia research. Unfortunately, the Intermedia system was only available on the AUX platform (a UNIX operating system for the Macintosh), which has severely hampered it to spread into real applications.

Intermedia was designed to demonstrate operating system developers that hypermedia functionality should be integrated into the desktop computing environment and was the first advocate for an open hypermedia system philosophy [Meyrowitz'89]. The technology was not available at that time, which explains why Intermedia is still to be considered a monolithic system.

Linking Protocol

In Intermedia, nodes are called 'documents' to be edited by editors part of the Intermedia system. The Intermedia editors represent standard desktop applications like word processors, drawing packages, e-mail, …. The Intermedia project coined the idea of an anchor (although it was Dexter that actually named it anchor, the very first Intermedia papers use the term block) to be an arbitrary selection within any readable document. According to the Intermedia philosophy, creating links should be as easy and as generic as the cut/copy/paste metaphor of desktop environments and thus must be supported by operations in the operating systems underlying those desktop environments. These operations are collected in a so-called linking protocol, a theme that returns in the discussion of open hypermedia systems.

Webs, or the Separation of Structure from Data

Another important concept in the design of Intermedia was that of a web. The basic assumption is that one document (i.e. a node in the generic hypertext terminology) can be part of different navigation networks and that the reader must be able to switch those navigation networks freely, depending on the reader's context. In Intermedia terminology, one such navigation network is called a web and the Intermedia system had special 'Web Views' containing automatically generated layouts of the network structure.

The reason why the web concept is so important, is that it implies that a hypermedia system should "get the links out of the data" or stated otherwise "hypermedia separates structure from data". This principle has proven so valuable that it is almost standard in the design of current hypermedia systems. Nevertheless, the world-wide web ignored this principle with its notion of embedded links, which explains most of the critique of the hypermedia community. We come back to this critique in the section on open hypermedia (p.22).

Object-Oriented Software Engineering

The final justification for discussing Intermedia is that the project advocated for the main theme of this dissertation: the cross-fertilisation between object-oriented software engineering and hypermedia. Intermedia [Meyrowitz'86] was actually a hypermedia specialisation of the fairly general MacApp framework [Schmucker'86] and promoted object-oriented databases as back-end for hypermedia storage [Smith,Zdonik'87].

Both NoteCards and Intermedia are stand alone system that store structures of information. However, they target different application domains and concerning other characteristics they are quite different. NoteCards was there to support individuals in structuring information, so the emphasis was on authoring and browsing tools. Intermedia was designed to support small groups working on shared material so here issues fine-grained concurrency control, tool integration were considered very important.

The research agendas in figure 1 give a glimpse of the problems with this generation of systems and most of them are tackled in the three streams of research that live in the hypermedia field today. But before discussing the issues and contributions of these three streams of research, we mention two miscellaneous hypermedia projects that play an important role in the remainder of this dissertation.

c) Miscellaneous — HyperCard and Dexter

This overview would be incomplete without discussing the HyperCard software [Apple'89] and the Dexter Hypertext Reference Model [Halasz,Schwartz'90]. We classify them in a miscellaneous category as neither of the two is to be considered a real hypermedia system.

HyperCard

HyperCard is to be considered the first really popular hypermedia product in the world and is certainly the first software product that introduced hypermedia to the broader public. There are several reasons for this popularity. First of all, it was bundled for free with every Macintosh computer, so all Macintosh users had access to it. Second, numerous 'stackware' applications were available for almost any problem domain, if not for free then certainly at very low prices. And finally, the product was designed so that anyone who could use a computer could read and create 'stacks'. (Note that more or less the same ideas are part of the world-wide web success story).

The Componentware Approach to Hypermedia

HyperCard is strongly based on a card metaphor. Nodes are called cards and are collected in a stack to form a single hypertext. Nodes take up a fixed size on the screen and can contain several widgets containing text, drawings, …. The behaviour of these widgets is controlled by means of scripts written in the HyperTalk scripting language. There is no explicit notion of links, but some HyperTalk statements can achieve hypermedia navigation effects (i.e. 'goto card …').

HyperCard must be mentioned in this overview because of its end-user tailorability. HyperCard really advanced the state of the art in hypermedia authoring tools, by offering end users an easy-to-use graphical programming environment including a powerful yet easy to learn scripting language. In that sense, HyperCard is actually an ancestor of the componentware approach for building software [Udell'94]: provide a library of powerful customisable components and let end-users assemble their own applications with it. Compared to object-oriented software engineering, the componentware approach is somehow the other side of a spectrum, hence we find it so important.

Is HyperCard a Hypermedia System ?

HyperCard is a good example of the implications of using a definition based on nodes and links like found in [Halasz'87], or a definition that emphasises on the style of building systems like the one we adopted from [Halasz'91].

Despite of its success (or maybe because of its success ?) some hypermedia researchers (see [FountainEtAl'90], [Mylonas,Heath'90]) doubt whether HyperCard is a full fledged hypermedia system[1], among others because it has no explicit notion of links. However, the definition we adopted classifies HyperCard as a hypermedia system, because it offers a degree of end-user customisability rarely met in other hypermedia software.

Nevertheless, because all HyperCard links are computed in scripts and because these scripts reside within the cards, the HyperCard design ignores the principle of "getting the links out of the data". Obeying this principle is not mandatory for being a hypermedia system, but it is definitely a help for providing extra navigational aids (see among others [Lai,Manber'91]).

The Dexter Hypertext Reference Model

The Dexter Hypertext Reference Model [Halasz,Schwartz'90] (Dexter for short) was the result of two small workshops on hypertext with representatives from many of the major existing hypermedia systems. The model attempted to capture the state of the art of that time's most prominent hypermedia systems (i.e. Augment, Document Examiner, IGD, FRESS, Intermedia, HyperCard, Hyperties, KMS/ZOG, Neptune/HAM, NoteCards, Sun Link Service and Textnet), systems belonging mainly to the generation of monolithic hypermedia systems. One of the original goals of the Dexter Hypertext Reference model was to define a hypermedia interchange standard for a wide range of existing and future hypermedia systems, and although this goal was never realised, the Dexter model had a considerable impact on subsequent hypermedia research; some hypermedia systems even initiated their design using the Dexter specification (notably DeVise hypermedia-DHM [Grønbaek,Trigg'94] and the Amsterdam Hypermedia Model-AHM [Hardman,Bulterman,VanRossum'94]).

The design of the Zypher framework was based on the Dexter Model as well, and we refer to Chapter 3 (Data Structures for an Interoperable Hypermedia Framework- p.76) for a more detailed discussion of the Dexter Model. Here we focus on the important ideas.

Separation of Concerns: Storage Layer, Within-Component Layer & Run-time Layer

First, by adapting the generic hypertext system architecture of the Hypertext Abstract Machine (HAM) model [Campbell,Goodman'87] Dexter emphasised the separation of concerns important in hypermedia system design. As depicted in figure 2, Dexter proposes three layers: the within-component layer (mostly referred to as the within layer), the storage layer and the run-time layer. We quote the original paper [Halasz,Schwartz90] to define the responsibilities of each layer.

1 According to [Nielsen'90], p. 93, Bill Atkinson (the designer of HyperCard) has admitted that it was not really designed as a hypertext product.

The storage layer "*models the basic node/link network structure that is the essence of hypertext. […] The storage layer focuses on the mechanisms by which link and non link components are 'glued together' to form hypertext networks. The components in this layer are treated as generic containers of data. No attempt is made to model any structure within the container*". The within-component layer "*is specifically concerned with the contents and structure within the components of the hypertext network*". The run-time layer provides "*tools for the user to access, view and manipulate the network structure*".

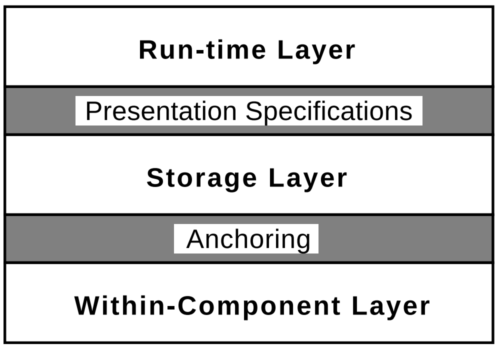

Figure 2: Dexter Model Layers

The emphasis of the Dexter model is on the storage layer and the two adjacent protocols. Using these two protocols, Dexter specifies a standard way of constructing hypermedia structures and storing them into an underlying "hyperbase". The within-component layer would rely on existing standards such as SGML and HyTime, while the run-time layer is the responsibility of any (hypermedia) application that adheres to the Dexter protocol.

Careful analysis of the definitions of the layers reveals that the separation of concerns between the storage and within-component layers is actually a rephrasement of the "get the links out of the data" principle. Also, the separation of concerns between the storage and run-time layer is actually a preparation for the open hypermedia philosophy, where any desktop application is allowed to manipulate the hypermedia navigation network.

The Anchor concept

Stating that the storage layer should be separated from the within-component layer is one thing, but then how should both layers interface with each other ? A hypertext system obeying the principle of "separating structure from data" requires a mechanism for addressing locations within that data. It was a major contribution of the Dexter model to formalise the concept of anchors (borrowed from the Intermedia project) for that purpose. Dexter also proposed the concept of presentation specifications as the glue between the storage layer and the presentation layer, but the hypermedia community did not pick up the idea.

We classified HyperCard an Dexter in a miscellaneous category, but again they target a different application domain and so we can expect some fundamental differences between the two as well. HyperCard aimed to support an individual in organising and presenting information, so the emphasis was on authoring and presentation. Dexter was mainly influenced by hypermedia systems supporting small groups of people working on shared material. Hence the emphasis on the storage layer, which among others enables to incorporate concurrency and version control.

d) The Open Hypermedia Systems — DHM and Microcosm

A first observation about the generation of monolithic hypermedia systems is that they are not 'open' in the sense that only applications part of the hypermedia system are able to process the information inside the navigation network. As a result, hypermedia systems are used for specialised tasks only; not in the daily information processing since people use other tools there (e-mail, text-processors, …) [Meyrowitz'89], [Brown'90].

The above observation gave rise to a stream of research generally referred to as open hypermedia. One of the best specifications of what open hypermedia research encompasses has been formulated in a scenario where engineers from the Boeing Airplane Manufacturing Company propose a hypermedia system as a way of integrating data, tools and services that support engineer's work practices [Malcolm,Poltrock,Shuler'91]. We come back to this scenario when we discuss open hypermedia (p.22). Here we discuss two representative examples of well-known open hypermedia Systems, and name most of the other known hypermedia systems.

Microcosm: an example of a link service raising the interoperability issue

Microcosm [FountainEtAl'90] is one of the first hypermedia systems to incorporate interoperability standards provided in the operating system (their first version used DDE - Dynamic Document Exchange under Windows) to allow external applications to make use of the linking facilities provided by the hypermedia system. Such an approach is typically called a link service (see also [Pearl'89]). Microcosm comes along with several built-in document viewers and the project investigated several ways to make external applications aware of the presence of the link service ([Hall,Hill,Davis'93], [DavisEtAl'92], [Davis,Knight,Hall'94]).

An interesting remark about the Microcosm project is that they have built up a notable reputation in dealing with multi-media archives. Among others by building electronic versions of the Mountbatten and Churchill archives. Noteworthy is also that, after a long period of experimentation in University labs, Microcosm has recently become a commercial product.

Generic Links and Filters

Generic links are an important ingredient of the Microcosm philosophy. The Microcosm project values links as an important store of knowledge besides the documents themselves. If the links are bound too closely to the documents, they cannot be applied to new documents. The similarities with the web concept in the Intermedia project are striking and indeed Microcosm is a strong advocate for the "get the links out of the data" principle [Davis'95].

Microcosm takes a slightly different approach however. Besides specific links between particular locations inside specific documents, the Microcosm system has this notion of a generic link, which once authored from a certain pattern (be it a text string or a piece of multi-media information [LewisEtAl.'96]) to a destination anchor, may be followed from any occurrence of the pattern in any document. Following generic links makes the system search the link base for all matches of a certain pattern to compute a list of possible target locations. Users can control this search by plugging filter modules, where each filter represents a special matching/searching algorithm.

The Componentware Approach to Open Hypermedia

Microcosm can be regarded as representing the componentware approach to open hypermedia. In essence, the Microcosm system is a link service for a collection of tailorable modules to be assembled in a hypermedia environment. There exist two types of modules: viewer applications and filters.

Microcosm classifies the viewers in three categories. Fully aware viewer applications [Hall,Hill,Davis'93] are viewers where the source code of the viewer was available and has been enhanced to include calls to the Microcosm API. The more interesting categories are the semi-aware and universal viewers [Davis,Knight,Hall'94]. Semi-aware viewers are third party applications that can be tailored (by means of a scripting language or macros) to include at least part of the Microcosm functionality. The generic example for these kind of viewers is Microsoft Word, which can be extended quite easily using the VisualBasic scripting language and the Microsoft Windows DDE inter-application communication facilities. The universal viewer is the means to handle applications that cannot otherwise be integrated with the Microcosm link service. It is a kind of a parasitical program (a "shim" in Microcosm terminology) that is wrapped around an unaware application and handles communication between the Microcosm link service and the application.

Filters are the modules that situate themselves on the other side of the link service. When a viewer application passes a message to the Microcosm link service, the message is passed through a chain of appropriate filters. Each filter decides what actions to take and returns messages to the link service which in turn passes them to the appropriate viewer applications.

Besides the tailoring facilities offered by viewer applications and filters, there is yet another tailorability dimension and that is the communication protocol between the different modules. There, Microcosm adheres to a free-format message passing protocol where modules communicate by exchanging messages containing lists of slots. The list of possible slots is unrestricted and receivers of the messages are not forced to interpret all the slots in order to understand the message. This idea is now taken as the basis for a definition of a standard communication protocol between open hypermedia systems. Below is shown how the current proposal [Davis,Lewis,Rizk'96] would request for opening a document called "OHP.htm" in read-only mode. The "DocumentNickName" slot is an example of an optional slot that can be used to provide redundant descriptions of the document.

```
\Subject LaunchDocument
\DocumentName "OHP.htm"
\ReadOnly True
\DocumentNickName "The OHP Protocol"
\DocumentType HTML
\DataCallBack False
\Channel 40
```

DHM (DeVise Hypermedia): The Framework Approach to Open Hypermedia

DHM [Grønbaek,Trigg'94] is an open hypermedia system designed to support co-operative design in (among others) large engineering projects. In contrast with Microcosm, DHM is more oriented towards object-oriented software engineering techniques to provide the kind of openness demanded in engineering environments.

As such, the DHM project incorporates object-oriented database and framework technology (again, we see the influence of the Intermedia project). Also, DHM allows end-users to extend the system by providing a kind of access to the source code level without the need to employ the full development environment [Grønbaek,Malhotra'94]. This is actually an improvement over the API approach taken by NoteCards or the framework approach taken by Intermedia.

An important distinction between DHM and Microcosm is that Microcosm enforces viewer applications to store their own documents, while DHM provides an extra service to the viewer applications by allowing them to store the documents inside the hypermedia system. The latter is very important in collaborative settings to handle integrity, version control and change notification.

DHM and Dexter

Besides being an open hypermedia system, DHM has an outstanding reputation concerning extensions to the Dexter Reference Model [Grønbaek,Trigg'94], [Grønbaek'94], [Grønbaek,Trigg'96].

Dexter was an attempt to capture the design of the generation of monolithichypermedia systems, but was underspecified to play its role as a reference model for exchange of hypermedia. DHM has proposed some extensions and modifications, among others concerning dangling links, link directionality, anchoring within composites [Grønbaek,Trigg'94]; locking and notification [GrønbaekEtAl.'94]; composites [Grønbaek'94]; location specifiers and reference specifiers to deal with embedded references, anchors in external documents and dynamic hypermedia structures

[Grønbaek,Trigg'96]. The DHM project has proven that the Dexter model —although a reference model for monolithic hypermedia systems— is a good basis to build open hypermedia systems.

Other Open Hypermedia Systems

Since open hypermedia is one of the main themes of this dissertation, we mention the other systems and projects that should be reckoned as open hypermedia. Sun's Link Service [Pearl'89] is a first attempt to incorporate a linking protocol in the operating system so that any application can make use of it. MultiCard is an open hypermedia system somewhere in between Microcosm and DHM. With Microcosm it emphasises on an extensible linking protocol that enables third party applications to access the link service; like DHM it offers hyperbase functionality with concurrency control features. HyperTED [Vanzyl'94] is an open hypermedia system that experiments with features specific to particular application domains to see whether this may help in solving open hypermedia problems. Hyperform [Wiil,Legget'92], [Wiil,Legget'93] and its successor Hyperdisco [Wiil,Legget'96] is much like DHM in the sense that is also based on object-oriented techniques to extend the system. One of the appealing features of Hyperform is the way it allows to experiment with various approaches for concurrency control and notification. The Hypermedia Research Laboratory in Texas developed a series of hyperbase and hypersystem prototypes called HB_{0-3}/SP_{0-3} [Kacmar,Legget'91], [Legget,Schnase'94] with the latest descendant named HOSS [NürnbergEtAl'96]. These prototypes stress the importance of processes in an open hypermedia system and lately started to explore potential benefits of having maximal operating system support for hypermedia functionality. Chimera [Anderson,Taylor,Whitehead'94] is —to our knowledge— the only system that demonstrates how third party applications can be used to enrich software development environments.

Microcosm and DHM are both open hypermedia systems. The reason they have chosen a different approach is because they aim to support a different application domains. Microcosm wants to support individuals working with various kinds of information, hence the emphasis on tools for authoring links and on customisability. DHM supports co-operative design in large engineering projects, so the emphasis is more on concurrency control, sharing of information and uniform tool enhancements.

e) The Distributed Hypermedia Systems — ABC and WWW

A second observation about the generation of monolithic hypermedia systems is the lack of 'distribution', in the sense that all the information inside the navigation network is stored in one centralised repository. In most cases, the centralised repository is a file system, although some hyperbases apply database technology as well (i.e. [Schütt,Streitz'90], [Wiil,Legget'92], [Wiil,Legget'93], [Wiil,Legget'96]). To some extent, the upcoming distributed database technology may serve the needs of distributed hypermedia. However, to implement the Xanadu vision, one must leave the notion of a centralised repository altogether, as some hypermedia systems do. The latter are kind are classified as the distributed hypermedia systems.

WWW: World-Wide Web

The hypermedia project coming nearest to implementing the Xanadu vision was the world-wide web project [Berners-LeeEtAl'94]. The world-wide web consists of an extensible set of "page-servers", where each page contains a number of embedded references to other pages residing on (often remote) servers. Web surfers (jargon for users reading world-wide web pages) employ browsers (the client programs) to navigate the loosely coupled graph of web pages by loading these pages and activating the embedded references to load referenced documents. The world-wide web utilises the Internet wide area network to connect all those servers, so that users really experience having the world at their fingertips: with a single mouse click they can jump to the other side of the world.

The world-wide web was developed at CERN as a medium to exchange scientific results, but rapidly reached momentum outside the scientific community and experienced an exponential growth of running servers and clients. Most software vendors, certainly the very large ones like Microsoft and ORACLE, adapt their strategies and software to incorporate some kind of web functionality.

The HTML, HTTP, URL Acronyms

There are a number of acronyms often used to refer to the world-wide web, but are in fact but technological cornerstones for enabling the establishment of such a large scale distributed hypermedia environment.

HTML (Hypertext Mark-up Language) is an interchange standard used to submit pages from the server to the clients. The idea is that an ordinary text document includes several "mark-ups" that tell the client how to format it. HTML includes mark-up for titles, boldfacing, graphics and —most important— anchors. Anchor mark-up contains the embedded references to the other pages and is supposed to be highlighted by client programs (typically in some colour and underline). HTML is defined as a growing standard, in the sense that future versions of the HTML standard are supposed to embody the older versions.

HTTP (Hypertext Transfer Protocol) is the network protocol most commonly used to transfer pages. These pages may be in HTML format, but HTTP can handle other formats as well (the HTTP protocol contains limited format negotiation). An important limitation (although essential on such scale) of the HTTP protocol is that it is stateless, i.e. the network connection is only held for the time needed to transfer the page. This makes it especially hard to use the world-wide web in collaborative settings. Note that most web browsers adopt other network protocols like FTP (file transfer), SMTP (electronic mail) as well.

URL (Universal Resource Locator) is the name for the format of a page address, which is used in the embedded references and anchor mark-up. The URL is actually an extensible addressing format, which makes it easy to refer to new types of page servers. A URL is prefixed with a key word identifying the address space of the referenced page (i.e. 'file' for file servers, 'http' for HTTP servers, 'ftp' for FTP servers, 'mailto' for electronic mail, …). The format of the rest of the address depends entirely on this prefix, but consists in most occasions of a set of identifiers separated by "/" (a hierarchical level) or "?" (an argument to a query).

Most world-wide web servers map HTTP requests to HTML documents residing on some file system. However, the HTTP protocol and the HTML format are sufficiently simple to be easily mimicked by other programs. For example, it is quite easy to set-up a program that translates user requests into database queries and returns the result in HTML format. Or a server may decide to store documents in another format and generate HTML on the fly with each request.

The diversity of protocols and formats and the ability to generate everything on the fly makes it hard to discuss the web as a hypermedia system. For example, the Web is frequently criticised as not respecting the "separate data from structure principle" (for using embedded references), while this is counter argued by saying that a server may very well store the links apart from the data and embed the links only when transferring the page (i.e. like in the Hyper-G [Andrews,Kappe,Maurer'95] and HOME [DuvalEtAl95] projects).

ABC: Artefact Based Collaboration

ABC is a project that has taken into account the issue of distribution from the very beginning. To deal with large scale hypermedia (such as in the aerospace industry

[Malcolm,Poltrock,Shuler'91]), "the architecture must permit distribution across multiple hardware platforms; to be distributable, the data model must be partitionable into objects that tend to be accessed separately and that have relatively few links and/or dependencies with other objects." [Smith,Smith'91]. Besides nodes and links, the ABC design is based on the notion of subgraphs, providing a natural way to partition the overall graph structure and distribute it over a number of graph servers. The ABC system comes with a multitude of network browsers that provide end-users with the necessary tools to manipulate the complex navigation structures.

The striking difference with the world-wide web project is the support for collaboration. The ABC project has investigated many issues related with sharing material over distributed servers, such as access control and concurrent access [Schackelford,Smith,Smith'93].

ABC is also an open hypermedia in the sense that the primitive nodes in the graph structure contain files to be processed by third party applications. ABC takes the viewpoint that hypermedia graph servers should become the file systems of the next generation of operating systems [Smith'96] and that hypermedia navigation will be an essential ingredient of the next generation operating systems (i.e. the graph browsers will replace the file managers of today's desktop environments). With the growing interest for world-wide web technology (among others by operating system vendors like Microsoft) this seems to be very promising.

Both being representatives of distributed hypermedia systems, the world-wide web and ABC are quite different with respect to the target application domain. The world-wide web aims to support individuals in browsing and publishing material and most work is on the development of interchange standards and communication protocols. ABC is about supporting groups, so incorporates various concurrency control mechanisms.

f) The Spatial Hypermedia Systems — Aquanet and HyperCafe

A third observation about the generation of monolithic hypermedia systems is the poor performance when dealing with 'dynamic' structures. Most hypermedia systems perform well for information presentation, but force authors to specify all navigation relationships explicitly by hand. So the support for structuring information can be improved, certainly when it comes to dynamic structures.

Aquanet and its successor **VIKI**

Although there were other systems that abandoned the basic node/link model to deal adequately with dynamic structures (i.e. String searching in SuperBook [Remde,Gomez,Landauer'87]; Petri nets in Trellis [Furuta,Stotts'89]; Taxonomies [Parunak'91]) it was with the Aquanet project [MarshallEtAl'91] that the Spatial Hypertext research took off.

The idea underlying the Aquanet project is to support a group in structuring collective knowledge by providing multiple views on the same knowledge structure. However, unlike traditional hypertext approaches, users are not forced to make these knowledge structures explicit by connecting nodes with typed (labelled) links (i.e. gIBIS [Conklin,Begeman'88]). In Aquanet, relationships are inferred from the spatial organisation of nodes [Marshall,Shipman'93]. VIKI [Marshall,Shipman,Coombs'94] and [Marshall,Shipman'95], which is the successor of Aquanet, allows to formalise those emerging structures on user's demand.

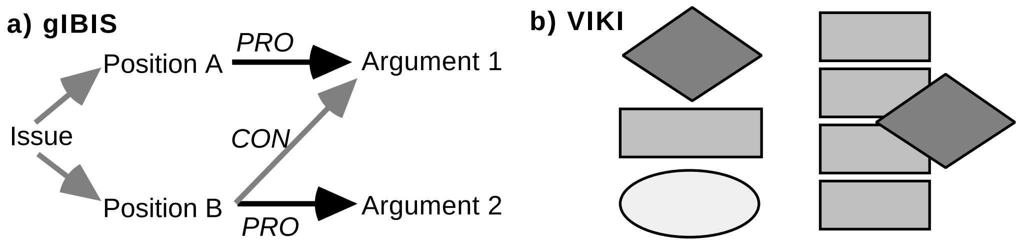

Figure 3: Traditional Hypertext (a) versus Spatial Hypertext (b)

Figure 3 illustrates the difference between a traditional way of creating a hypertext structure and the spatial way. On the left we see part of a gIBIS network stating that there are two possible positions for a given issue and that there are two arguments sustaining the positions; moreover one argument not only sustains one position but also counters the other position. The create such an argumentation structure, a user creates five nodes (one issue, two positions and two arguments) and five links. Note that creating a link is a complex operation, one must choose the type of the link, mark its source and target an provide an optional label.

To the right of figure 3, we see two VIKI structures: one with a diamond, rectangle and oval denotes a composite relation; the other with four rectangles and a diamond denotes a labelled relation. This means that to create a composite relation, a user must place three node with three different types one beneath the other. And if a user places a group of nodes with same type one beneath the other, and an extra node with different type on top of it this is recognised as a labelled relation. So, creating relationships with VIKI requires less steps than traditional hypertext approaches.

HyperCafe

Aquanet and VIKI emphasise the idea of structuring knowledge and somehow lose the idea of navigation and multi-media. The HyperCafe project [Sawhney,Balcom,Smith'96][2] reincorporates both notions into the spatial hypertext research. HyperCafe is designed as a cinematic experience of hyper-linked video scenes; the metaphor is that of a human visiting a cafe and picking parts of the stories told by all the other visitors. While the users follow a story in the foreground of the video, the background shows characters telling other stories to each other. Users can select those background characters to switch stories. So far, this is navigation in video, but the trick is that users can choose to run multiple foreground stories by arranging several simultaneously running videos on the screen. Such spatial organisation produces a whole new story, hence the classification as spatial hypermedia.

Why is this experiment worth mentioning ? Remember that, according to our definition, hypermedia is a "style" rather than a number of features and that style is about freedom - about customisability. However, there is not so much freedom for "readers" of traditional hypermedia, as they can only follow the links provided by the authors. HyperCafe shows an extra degree of freedom, where users can create stories that were not intended by the authors.

Again we see that the differences between the two systems stem from the target application domain. The Aquanet and VIKI design are based on the "WYSIWID" principle, which is an acronym for "What You See Is What I Did". The systems supports co-operation and the reason they want easy authoring is because they want to support discussion in little groups. HyperCafe maximises the customisability of individuals and there the emphasis is on the presentation of information.

2 This paper was rewarded with the first Douglas Engelbart award at the Hypertext'96 Conference [HT'96].

Conclusion

The purpose of this section was to confront the reader with the important ideas that live (and lived) in the hypertext and hypermedia community. The section included a definition of hypermedia as a combination of navigation and a style. Navigation is about browsing non linear information structures. Style emphasises on a feeling of freedom, in the sense that users must be able to customise the presentation of information in some way. The definition comes with some basic vocabulary (i.e. node, link and anchor).

The important ideas were illustrated with an overview of existing hypermedia systems, roughly classified as pioneering, monolithic, miscellaneous, open, distributed and spatial systems. The overview emphasised two threads important to the remainder of the dissertation: tailorability and the correct separation of concerns. The latter stresses the "get the links out of the data" principle and was illustrated with the Dexter hypertext reference model. The former was motivated by the observation that each hypermedia is designed for another application domain and was presented as a spectrum ranging from support for individuals as opposed to groups and componentware as opposed to object-oriented software engineering.

Open Hypermedia

While the previous section provides us with a general introduction of hypermedia, this section discusses the particular subfield called open hypermedia systems. Open hypermedia research starts from the assumption that to provide real support for daily information processing, hypermedia systems must integrate external applications to manipulate information. Several compelling issues pop up when dealing with this integration requirement.

Open hypermedia research is quite an active subfield. Since 1989, the hypermedia conferences accepted several papers discussing approaches for open hypermedia and already three workshops were organised to exchange ideas (i.e. the Konstanz workshop [Aßfalg'94]; the 1st Workshop on Open Hypermedia Systems [Wiil,Østerbye'94]; the 2nd Workshop on Open Hypermedia Systems [Wiil,Demeyer'96]).

As always within an active field of research, it is quite hard to give a precise definition. Here we present different approaches, which should give a good overview of what researchers mean when speaking of open hypermedia systems. Several authors have attempted to define an open hypermedia system, but most definitions fall too short and are supplemented with requirement lists. As such requirement lists tend to be vague and imprecise, there is a growing tendency to write scenarios describing how these requirements fit into the usage of an open hypermedia system for the daily work of a group of people performing a particular task. Finally, we present the Flag taxonomy as a first attempt to classify open hypermedia approaches rather than systems.

Open Hypermedia Definition & Requirements

We adopt the most recent definition of an open hypermedia system, as we agree that it gives a reasonable summary of current thinking.

"*An open hypermedia system is a hypermedia system, where the term open implies the possibility of importing new objects into a system. A truly open hypermedia system should be open with regard to:*

1. *Size: It should be possible to import new nodes, links, anchors and other hypermedia objects without any limitation to the size of the objects or to the maximum number of such objects that the system may contain, being imposed by the hypermedia system.*
2. *Data Formats: The system should allow the import and use of any data format, including temporal media.*
3. *Applications: The system should allow any application to access the link service in order to participate in the hypermedia functionality.*
4. *Data Models: The hypermedia system should not impose a single view of what constitutes a hypermedia data model, but should be configurable and extensible so that new hypermedia data models may be incorporated. It should thus be possible to interoperate with external hypermedia systems, and to exchange data with external systems.*
5. *Platforms: It should be possible to implement the system on multiple distributed platforms.*

6. *Users: The system must support multiple users, and allow each user to maintain their own private view of the objects in the system.*" [Davis,Lewis,Rizk'96].

We elaborate on the requirement list part of the definition by combining the following sources[3]: the 7-issues research agendas [Halasz'87] and [Halasz'91]; the Boeing paper [Malcolm,Poltrock,Shuler'91], reiterated in [Poltrock'96]; a paper from the HB/SP series of systems [Legget,Schnase'94]; the preamble to the Konstanz Open Hypermedia Workshop Proceedings [Aßfalg'94]; papers from the Microcosm group [DavisEtAl'92] and [Hill,Hall'94]; a paper from the HyperTED project [Vanzyl'94]; a paper from the Hyperdisco project [Wiil,Legget'96] and finally —the one that is part of our open hypermedia definition and serves as the backbone— the paper on the Open Hypermedia Protocol [Davis,Lewis,Rizk'96].

OHS-REQ 1) Size

It should be possible to import new nodes, links, anchors and other hypermedia objects without any limitation to the size of the objects or to the maximum number of such objects that the system may contain, being imposed by the hypermedia system.

The size issue covers two different aspects: the size of an individual object and the number of different objects in the system. The former is a typical problem when dealing with multi-media data. The latter is crucial when dealing with large scale hypermedia systems (i.e. Boeing speaks in the order of 1 million nodes and 10 million links) or distributed systems (i.e. the world-wide web).

This issue of size has some important implications. First of all, size implies the possibility to access data residing outside the hypermedia system, since large object sizes may prevent copying the data into the system. Secondly, size affects scalability and thus addressability. A system having unlimited capacity needs at least an unlimited and scalable address space. Furthermore, as one cannot expect to know the sources of external data in advance, the address space must be extensible.

Storing information outside the hypermedia system challenges the design of the hypermedia system with respect to consistency. For example, one cannot ensure link consistency since data can be edited without informing the hypermedia system (this is called the editing problem). Link robustness demands for techniques and heuristics to fix corrupted link structures. Also, data residing outside the hypermedia system implies that the original data must remain unchanged (this is called document sanctity), since one cannot be sure to have write-access to the information (i.e. the data can reside on read-only media such as a CD-ROM). Document sanctity is another argument for the "get the links out of the data" principle [Davis'95].

OHS-REQ 2) Data Formats

The system should allow the import and use of any data format, including temporal media.

This can be read as a multi-media requirement, although it covers issues like legacy data formats and encryption techniques as well. The requirement implies at least the ability to launch another application that knows how to interpret the data format, since it is impossible to build a single hypermedia system that knows how to interpret all current and future data formats. Standards (see OHS-REQ 4) make life easier of course.

An implication not immediately visible from this requirement (but follows from combining it with part of OHS-REQ 4, exchange data with external systems) is that a

[3] The original idea to combine open hypermedia requirement lists came from Helen Ashman in a draft paper investigating how well the World-Wide Web performs on each of these requirements. As may be surprising at first, for many of the requirements the Web performs better than the proclaimed open hypermedia systems.

hypermedia system should be able to deal with virtual data as well. Virtual data is not available from some persistent storage but is generated by some external device or program; typical examples are video conferencing, database queries and simulation tools. Note that both temporal and virtual data impose special problems on hypermedia systems, like how can one specify anchors inside temporal or virtual data.

OHS-REQ 3) Applications

The system should allow any application to access the link service in order to participate in the hypermedia functionality.

No doubt this is a mandatory requirement for an open hypermedia system, raising the interoperability issue (i.e. making applications work together without explicit knowledge of each other). Lots of techniques are available to extend "any" application to incorporate hypermedia functionality, among others described in the HB_0/SP_0 (or Proxhy) paper ([Kacmar,Legget'91]), the Microcosm papers ([DavisEtAl'92], [Hall,Hill,Davis'93], [Davis,Knight,Hall'94]) and the HyperTED paper ([Vanzyl'94]).

This requirement states that an open hypermedia system should have a link protocol, accessible by any other application that wants to use it. This is the opposite of having the hypermedia system launch another application with a given set of data (as implied by OHS-REQ 2).

OHS-REQ 4) Data Models

The hypermedia system should not impose a single view of what constitutes a hypermedia data model, but should be configurable and extensible so that new hypermedia data models may be incorporated. It should thus be possible to interoperate with external hypermedia systems, and to exchange data with external systems.

Given the enormous variety of data models existing in the hypermedia field, this requirement is very hard to realise (conceive an open hypermedia system that is able to deal with the basic node-link model as well as a data model supporting spatial hypertexts). What makes it especially difficult is that lots of the information is formed by the "link" structure, so that an open hypermedia system must be able to incorporate different link models. Hence the need for an extensible link engine, which is free to choose the linking model that best suits the needs of the information structure it represents. Microcosm (with its generic links and filters) and DHM (with its location and reference specifiers) provide building blocks for such an extensible link engine.

A first related issue is that of standards: with a standard hypermedia reference model one could exchange data with other hypermedia systems without having to write translators for all possible combinations. Note that [Legget,Schnase'94] reports that a possible disadvantage of standards is the loss of structure. As far as data formats concern, hypermedia can rely on existing standardisation efforts such as HyTime [Newcomb,Kipp,Newcomb'91]. For the communication aspects, the open hypermedia community is developing a standard communication protocol for link services [Davis,Lewis,Rizk'96].

The second issue is that of extensibility, which brings us back to the style aspect in our hypertext and hypermedia definition (p.08). As hypermedia style emphasises a feeling of freedom and a degree of customisability, extensibility plays a crucial role. This is not a problem of the hypermedia field alone and there are lots of techniques available that can be applied in various combinations.

- Configuration tools: Special purpose extension tools, most of them with user-friendly graphical interfaces. Examples are the "Preferences" tools coming with most of today's desktop applications and the control panels in the Macintosh and Windows operating systems. In the hypermedia community, HyperCard is still the most prominent example of how configurable hypermedia software can be.

- Scripting languages & macro facilities: Programming languages aiming at skilled end users. Examples are HyperTalk (HyperCard's scripting language) and VisualBasic (the scripting language coming with the Microsoft Office suite of products).
- Dedicated programming languages: Full fledged programming language dedicated to extend one type of systems. Examples: Shell languages for operating systems; SQL; Java [Java'95]; MC2000 [Rizk,Sauter'92].
- General purpose programming languages: A full fledged programming language used for other purposes than extending a particular system. Example: Lisp in AutoCAD and GNU Emacs; Scheme in Hyperform [Wiil,Legget'92].
- Object-oriented frameworks: A reusable design for a specific problem domain implemented in some object-oriented medium and extended by plugging new objects. Example: Intermedia [Meyrowitz'86] and DHM [Grønbaek,Trigg'94].
- Open Implementation: A computational system with the ability to manipulate its own computations. Examples: Silica [Rao'91]; ApplFLab [SteyaertEtAl'94] & [SteyaertEtAl'95].
- Extensible data models: Systems with a data model that can easily be extended for certain purposes. Examples: filters in Microcosm [DavisEtAl'92]; hyperdocuments in [DeBra,Houben,Kornatzky'92].

OHS-REQ 5) Platforms

It should be possible to implement the system on multiple distributed platforms.

Depending on the scale of distribution, systems must incorporate a range of techniques to ensure fast response times (i.e. data caching, prefetching). To produce adequate results the system must gather knowledge about often followed navigation trails, usage patterns, ...

Also, distributed hypermedia systems raise the document sanctity issue and the editing problem (see OHS-REQ 1). Yet, the scope of possible solutions is completely different as consistency seems to be less crucial in large scale distributed systems such as the world-wide web. One may adopt some local consistency scheme complemented with periodic "garbage collection".

Supporting multiple platforms raises the issue of portability across different soft- and hardware platforms, an issue that overlaps with the data formats (OHS-REQ 2), applications (OHS-REQ 3) and data models (OHS-REQ 4) requirements. Indeed, there is a multitude of data formats, applications and data models and the choosing the best depends upon the intended soft- and hardware platform. One way to deal with portability across different soft- and hardware platforms is to incorporate some form of protocol negotiation, where part of the communication between modules is about choosing the optimal configuration.

OHS-REQ 6) Users

The system must support multiple users, and allow each user to maintain their own private view of the objects in the system.

Hypermedia has always played the role of enabling technology for computer supported co-operative work. This requirement handles two aspects related to this role: one of information sharing and one of information perspectives.

If multiple users work with the same hypermedia structure, then the hypermedia system must control access to the shared data. Authority control is the process that verifies whether a user is allowed to read or modify data. Generally, an open hypermedia system should not have an artificial distinction between authors (allowed to modify) and readers (read only access), such as imposed by systems having a separate authoring environment. Concurrency control deals with the problems of simultaneous access to the same data element and is essentially based on locking (restrict access temporarily to ensure safe data transfer) and notification (warn the other users working with a data element that something has happened).

An important aspect of co-operative work is the ability to create private work spaces. In a hypermedia context this means that (a subgroup of) users must be able to build a personal substructure inside the global hypermedia structure. This is again a plea for the "get the links out of the data" principle, since a shared node may participate in different link structures. This aspect interferes heavily with the linking models supported in the hypermedia structure.

Closely related is the issue of version control (the ability to have several variants of the same data element and/or structure). Many hypermedia researchers agree that versioning is important, yet few systems provide version control and few users seem to complain[4].

The above requirement list makes it very clear that incorporating hypermedia functionality into the everyday working environment involves more than providing link services to third party applications. The closer examination of the items in the list reveals that open hypermedia systems cover themes like interoperability (1-3), extensibility (4) and distribution (5-6), which are central themes in the design of all kinds of open systems — operating systems (i.e. UNIX and OS/2), databases (Exodus and CORBA), inter application communication (OLE, OpenDoc), tailorable software (i.e. Emacs and AutoCAD), programming languages (Smalltalk and CLOS). By adopting this requirement list as the definition for an open hypermedia system, we see that open hypermedia systems are quite representative for the domain of open systems. *Which makes open hypermedia systems an excellent choice for experimenting with techniques to build open systems*, a central corner stone in our argumentation.

Open Hypermedia Scenarios

From its conception (among others in [Yankelovich,Meyrowitz,VanDam'85]), open hypermedia research was aiming to get hypermedia functionality out of the hypermedia systems and into the every day environments people use for working with information. With the "Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise" [Malcolm,Poltrock,Shuler'91] paper, the hypermedia community finally distilled a requirement list out of the working practices of engineers participating in the design and manufacturing process of airplanes. The paper is a milestone in hypermedia research because (quoting Frank Halasz [Halasz'91]) "*a user-centered technology like hypermedia needs users to drive the research*". As the authors of the paper were all associated with Boeing —world's largest producer of commercial airplanes and one of the biggest employers in the US— this could count as a user community.

Another reason why the paper is so appealing, is because it contains a scenario illustrating how the requirement list fits into the everyday working practice of an engineering team. The scenario described how a new team member could learn about a project (navigating through the project notebook), exchange information with other team members (e-mail and tele conferencing), and contribute to the project's goals (preparing reliability analysis) all within an environment seamlessly integrating all the different tools applied by engineers in their day to day work. The open hypermedia community has picked up this scenario idea: it reappeared in [GrønbaekEtAl.'94], [Legget,Schnase'94] and [Davis,Knight,Hall'94] and to some degree in (the presentation accompanying) [Vanzyl'94]. Moreover, the 2nd Workshop on Open Hypermedia Systems [Wiil,Demeyer'96] decided to set-up a web-site collecting and exchanging scenarios for open hypermedia usage (see http://www.csdl.tamu.edu/ohs/).

4 Version control is not part of the Boeing requirement lists [Malcolm,Poltrock,Shuler'91] and [Poltrock'96].

The Flag Taxonomy of Open Hypermedia Systems

Although the open hypermedia definition gives a good summary of current thinking, it is too requirement oriented and thus too long. Moreover, it is too strong, since none of the existing open hypermedia systems meet all of the listed requirements. The Flag taxonomy [Østerbye,Wiil'96][5] takes a completely different angle by making an attempt to capture the essence of an open hypermedia system and use them to classify open hypermedia systems.

The Flag taxonomy started of as a "written before the facts" conclusion for the first open hypermedia workshop [Wiil,Østerbye'94]. The workshop organisers prepared a set of slides with a number of diagrams showing many of the design trade-offs involved in the construction of an open hypermedia system. These diagrams were discussed during the workshop and although the preformulated conclusion proved incorrect, it served as an adequate framework for discussing open hypermedia systems. As, the workshop proclaimed the need for an open hypermedia reference model —i.e. much like the Dexter model was for the generation of monolithic hypermedia systems— these diagrams were seen as a good starting point. The flag taxonomy is actually a mixture of those concluding slides with some of the ideas of the Dexter model.

If one looks at the layered architecture of the Dexter model (see left hand side of figure 4) from an open hypermedia perspective, an important difficulty is revealed: the within-component layer is separated from the run-time layer by means of a storage layer. This is a typical symptom for monolithic systems, where a hypermedia system somehow constrains the possible contents of nodes to be presented in the run-time layer, because these node-contents must be stored inside the hypermedia system. However, this disregards a fundamental principle of open hypermedia systems where viewer applications need to store the node contents outside the hypermedia system. So, the Dexter architecture actually violates requirements 1-3 of the open hypermedia definition. To cope with this difficulty one can adapt the Dexter model and pull together the within-component layer and run-time layer. Doing so results in the "Dexter-pie" depicted in the right hand side of figure 4.

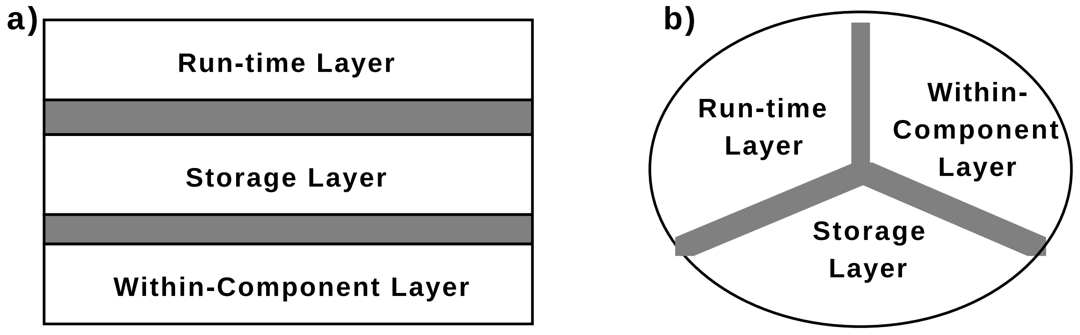

Figure 4: Layered Dexter Architecture (a) versus the Dexter Pie (b)

Remember that there is another important principle in the design of (open) hypermedia systems, which is the "get the links out of the data". In the Dexter model, this principle is captured in the protocols between the layers and this is inherited by the Dexter pie. The flag[6] taxonomy goes one step further, as it captures those two essential separations of concerns for an open hypermedia system by performing one additional cut (see figure 5). The horizontal cut separates contents from structure (i.e. data from links) while the vertical cut separates storage from run-time. This results in four functional modules, which the authors named storage manager, viewer, data model manager and session manager. Also there are four protocols between functional modules. Note that the protocol of prime interest in open

5 This paper was one of the nominees for the first Douglas Engelbart award at the Hypertext'96 Conference [HT'96].

6 A glance at Figure 4 combined with the Danish nationality of the authors reveals the inspiration for the name.

hypermedia research is the linking protocol between the viewer and the session manager (see [Davis,Lewis,Rizk'96]).

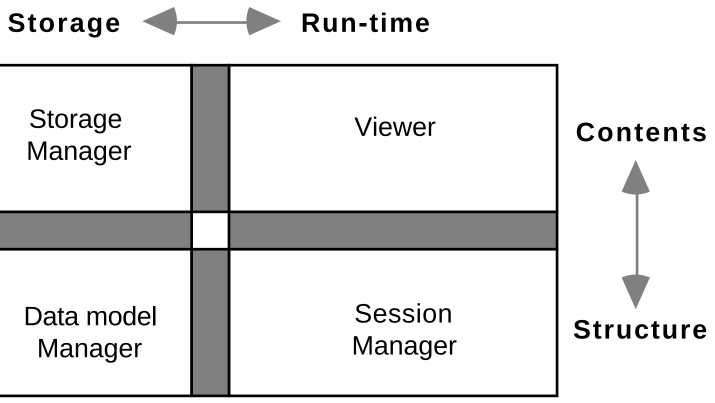

Figure 5: The Flag Taxonomy

What makes the flag taxonomy so appealing is that one can move the different configurations of the four functional modules and protocols and that most (if not all) configurations can be assigned a meaningful semantics corresponding with real systems. So the flag taxonomy enables a rough classification of hypermedia systems.

Such is depicted in figure 6, where a white region represents a unit in the hypermedia system that is a union of one or more functional modules in the flag. A greyed region stands for some protocol that exist between two modules and is accessible from outside. A black region represents a boundary between modules where no communication is allowed.

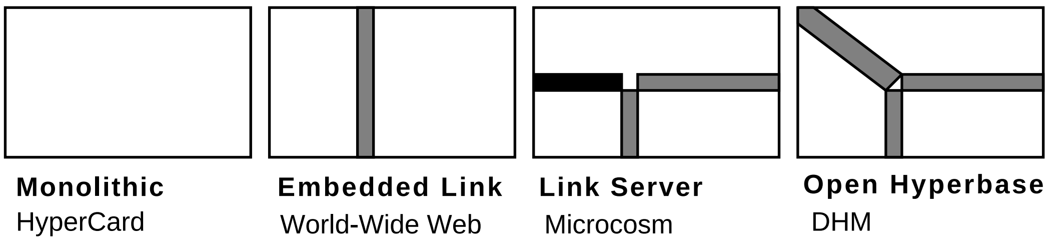

Figure 6: Some Meaningful Configurations for the Flag

A monolithic hypermedia system is a system where outside the hypermedia system one cannot distinguish the four functional modules (i.e. HyperCard) and is represented as a solid white rectangle. Embedded link systems like the plain vanilla world-wide web make a distinction between storage (i.e. a HTTP server) and run-time presentation (i.e. any web browser) but mixes structure with contents. Hence the vertical protocol between storage and run-time. A link server like Microcosm enforces third party applications to store and present their own data and as far as the hypermedia system concerns, there is no distinction between the storage and representation of data. However, third party applications can freely access the link services (horizontal grey protocol) using the linking protocol. Microcosm provides partial control over the storage and retrieval of links via the filter concept (vertical grey protocol) but this is not directly accessible for third party applications (horizontal black protocol). Open hyperbases have an extra service over link servers by offering viewer applications the possibility to store their data inside or outside the hypermedia system (the diagonal grey protocol).

The flag taxonomy can be used to separate open hypermedia systems from other hypermedia systems. The link server and open hyperbases are considered open hypermedia systems, as they do not impose a special data model on the viewer applications. The embedded link approach is not considered open, as viewer applications must adhere to the structure and contents format. According to the flag model, to be an open hypermedia system, there should exist a linking protocol between the session manager and viewer application. Note that the flag taxonomy represents the hypermedia system developer's point of view, which is opposed to the software system developer's point of view. In the latter, all systems with at least one protocol are open.

Although the flag taxonomy captures the essence of open hypermedia systems (i.e. separating storage from presentation and structure from contents) in an appealing metaphor, some important issues are not covered in the taxonomy. Only half of the requirements of the open hypermedia system definition are covered in the flag taxonomy; that is requirements 1 to 4 (size, data formats, applications) but not requirements 4 to 6 (data models, platforms and users). So the flag taxonomy covers only the three requirements that deal with interoperability, but not the ones that are about extensibility and distribution. However, one of the factors for adopting the requirement list was precisely that it covered the three main themes found in all open systems — interoperability, extensibility and distribution. So, we accept the design guidelines forwarded by the flag taxonomy (i.e. separate storage from presentation, separate contents from structure) but to study open hypermedia systems as a representative for other kinds of open systems, we need a more generic model. This more generic model is presented in the next section under the name of the Zypher perspective.

Conclusion

Open hypermedia is an active research area, so it is difficult to provide a precise picture of the field. To cope with this problem, we have presented three perspectives common in the field today.

The first perspective is presented as a scenario of an imaginary system for supporting the working practices of engineers participating in the design and manufacturing process of airplanes. The second —closely related— perspective is presented as a definition and an extensive requirement list. We have examined the items in the list, have sketched a number of issues and have pointed at common themes. An important observation for the remainder of the dissertation was that the requirement list covered important themes relevant in other kinds of open systems — i.e., interoperability, extensibility and distribution. This shows that open hypermedia is an excellent choice for experimenting with techniques to build open systems. The third perspective is presented as a taxonomy, which is a first attempt to summarise the important design principles for open hypermedia systems. The taxonomy forwards two essential separation of concerns in the design of open hypermedia systems, i.e. contents from structure and presentation from storage.

The Zypher Perspective

This section introduces our personal perspective on open hypermedia systems. We present this perspective because we feel that what is currently available lacks some important aspects on open hypermedia systems.

Reviewing our hypertext & hypermedia definition (p.08), we notice the importance of the word "style", which is about freedom, about customisability. Also, the open hypermedia definition (p.22) regards openness as the ability to incorporate new objects into as system, which is about adaptability. Although, style and adaptability are implicitly part of the requirement lists and scenarios, we want to make them an explicit element of our perspective since we want to focus on techniques to construct customisable and adaptable systems. Therefore, we present tailorability as a characteristic that encompasses both style and adaptability.

With respect to the flag taxonomy, we see the Zypher perspective as "another side of the coin". There are two arguments sustaining the position that the other side of this coin is indeed important. First of all, there is the way the flag taxonomy deals with the world-wide web. According to the flag taxonomy (see figure 6), the world-wide web is not an open hypermedia system because it violates the "get the links out of the data" principle. This may be true from a hypermedia system developer's point of view, but with its extensible architecture system developers would certainly classify it open. Secondly, there's is the way the flag taxonomy differentiates between Microcosm and DHM. Looking at figure 6, the flag taxonomy focuses on the way open hypermedia systems like Microcosm and DHM deal with the storage of documents. DHM provides hyperbase services, i.e. it allows third party applications to store the contents of the documents inside the DHM database, while Microcosm does not. We agree that this is an important difference, yet we see the difference between the componentware and framework approach (see "The Open Hypermedia Systems — DHM and Microcosm]" - p.15) at least as important. This difference is not covered in the flag taxonomy and yet it is important, also from a hypermedia system developer's point of view.

We propose tailorability as an encompassing view on openness. That is, we see tailorability as a way to achieve the kind of customisability needed to accomplish a true hypermedia style, but also as a way to capture the extensibility of the world-wide web, Microcosm or DHM. More importantly, we propose tailorability as a way to achieve openness in all kinds of open systems, such as operating systems, databases, inter application communication, tailorable software and programming languages. To support such an encompassing view, we need to be more precise on what is meant by the term tailorability. We define tailorability based on the notion of a design space as a generic abstraction of all kinds of open systems and identify three levels of tailorability afterwards.

The Zypher Design Space

Summarising the section on "hypertext & hypermedia" (p.07), we define a hypermedia system as a unique combination of navigation, storage and presentation technology. This summary is named the Zypher design space and it is shown in the left hand side of figure 7. The Zypher design space has three dimensions, known as the storage axis (enumerating all possible information repositories), the presentation axis (enumerating all possible viewer applications) and the navigation axis (enumerating all techniques to specify navigation relationships). A particular hypermedia system is then a relation (in the mathematical sense of the word) between points on these three axes and is represented by a 3-dimensional volume in the design space; for the sake of simplicity we represent such a volume as a cube. As announced earlier, an open hypermedia system is then a tailorable volume in that design space (see the right hand side of figure 7).

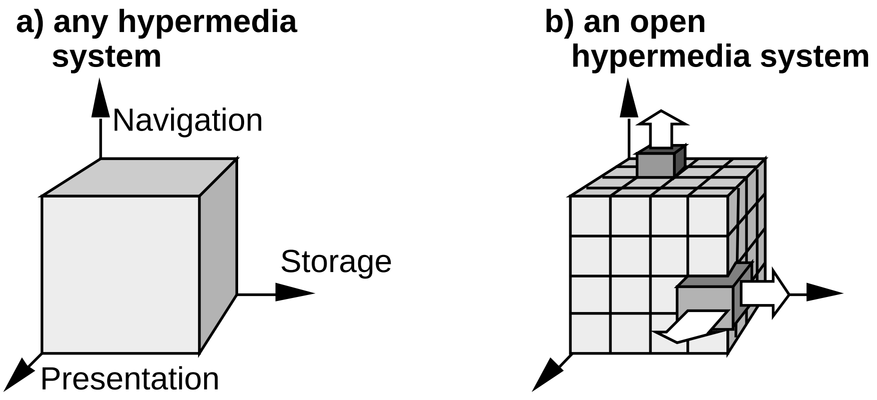

Figure 7: The Zypher Design Space

A very important remark is the choice of these axes is left open: the remainder of the argumentation is completely independent of the precise number and semantics of the axes. For our experiments we found those three valuable and we motivate their existence in the second part of this dissertation (see interoperability - p.78; navigation - p.86). However, a designer of an open hypermedia system —or any other open system for that matter— is allowed to change the axis system freely. The independence of the number and semantics of axes in the system is crucially important, as it is this aspect that allows us to generalise Zypher design space to any kind of other system. We come back on this issue when discussing the Zypher contribution (p.63).

Three Tailorability Levels

What is tailorability ? To a large degree, this question was answered during the discussion of the extensibility issue in the data model requirement (see OHS-REQ 4), where we list a number of techniques that can be used to adapt a hypermedia system (configuration tools, scripting languages & macro facilities, dedicated programming languages, general purpose programming languages, object-oriented frameworks, open implementations, extensible data models). This answer is not a satisfying one, as it emphasises on the techniques.

We propose a more analytical approach by presenting three levels of tailorability. Each level of tailorability is defined using the axes system of the Zypher design space in figure 7. Note the important property that *the definition of the three levels of tailorability is independent of the type and numbers of axes in the design space.*

Domain Level Tailorability

Domain level tailorability is the kind of tailorability needed to deliver a hypermedia system for a specific application domain. Typical usage of domain level tailorability in hypermedia systems would involve the incorporation of new data formats, extra information presentation methods or supplementary types of navigation techniques. *Domain level tailorability corresponds with the addition of extra points (or the modification of existing points) on the axes of the Zypher design space.*

System Level Tailorability

System level tailorability aims to deliver services that affect the global behaviour of the hypermedia system. Examples are concurrency control (i.e. manage concurrent access to shared data), logging (i.e. maintaining a log of navigation activities to provide backtracking features), caching (predict future behaviour on the basis of registered activities), authority control (check whether the user of the system has the privileges to see or modify information) and integrity control (control operations to preserve the consistency of the system's data structures). All these examples have in common that they change the behaviour of several modules in the system in a uniform way. *System level tailorability corresponds with wrapping uniform behaviour around different points residing on an axis of the Zypher design space.*

Configuration Level Tailorability

Configuration level tailorability aims to provide a 'plug and play' hypermedia system, where the coordination between the system modules is adapted without changing their internal implementation. Typical usage of configuration level tailorability is to make the system run-time extensible. For instance, the table of helper applications in most world-wide web browsers, where users can associate a viewer application with a given document type is about the configuration between the storage and presentation axis. The Java approach [Java'95] is another example, where world-wide web browsers load system modules for handling unknown document formats. The final example is protocol negotiation, where system modules communicate to choose the optimal protocol for exchanging data; protocol negotiation is valuable in cross-platform communication. *Configuration level tailorability corresponds with changing the relationship (in the mathematical sense of the word) between the points on the axes in the Zypher design space.*

The Puppet Master Metaphor

These three levels of tailorability play a crucial role in this dissertation. To make sure the reader recognises the differences and associates them with the appropriate level, we have devised the so-called puppet master metaphor. From this metaphor we have inferred three icons that are repeated throughout the text as little memory aids. The metaphor is based on the analogy between adapting a software system to a changing environment and adapting puppets to a new puppet play.

Domain Level Tailorability

When preparing a story, the puppet master conceives a number of puppets playing different characters. Archetypal puppet characters are the harlequin and the pierrot, where the former wears a costume with much coloured patchwork and a smiling face and the latter is dressed in white and holds a tear under the eye. To distinguish these characters the puppet master paints the puppet faces and dresses them up with different costumes.

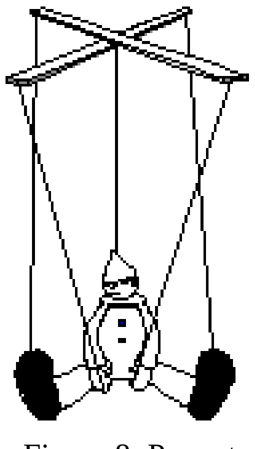
Figure 8: Puppet Master Metaphor

System Level Tailorability

However, for certain kinds of stories, some puppets require special abilities not captured by the basic puppet design. Take the example of a story that has a sword fighting scene. Pasting a sword onto the hands of an existing puppet is not enough, since a realistic fighting scene requires the ability to control the movements of the sword as well. Such a scene demands for an extra string to be attached to manipulate the special behaviour.

Configuration Level Tailorability

Puppet plays require the puppet master to co-ordinate the behaviour of each individual puppet. Some puppets need limbs that can be moved around freely, while other puppets require them to go in opposite directions. One must adapt the wooden cross co-ordinating the movements to achieve this kind of tailorability.

Note that the users of the system correspond with the audience watching the puppet: they are not supposed to know how the puppet is manipulated to produce the desired scenes in the play. However, just like the audience can influence the play by applauding and shouting, users can influence the behaviour of the system by setting preferences. The puppet master (i.e. the software engineer) then applies the tools on the appropriate level of tailorability to satisfy the audience.

Appropriateness of the Three Tailorability Levels

Having identified tailorability as a distinctive characteristic of open hypermedia systems and having proposed three levels of tailorability to handle different kinds of flexibility, the natural question that follows is how far these tailorability levels correspond with existing open hypermedia research ? To answer that question, we contrast two representative open hypermedia systems (Microcosm representing the componentware approach and DHM representing the framework approach) on each level and refer to the open hypermedia definition & requirements (p.22) to sustain our arguments.

Domain Level Tailorability

All open hypermedia systems satisfying the "size", "data formats" and "applications" requirements (see OHS-REQ 1, OHS-REQ 2 and OHS-REQ 3) are domain level tailorable, because satisfying these requirement boils down to lengthening the storage or presentation axes of the Zypher design space. For example, DHM has been tailored towards the domain of co-operative engineering (see [GrønbaekEtAl.'94]) by incorporating different viewer applications demanded by the engineers. Microcosm incorporated different multi-media viewers to deal with the Mountbatten and Churchill archives (see [FountainEtAl'90]). While discussing OHS-REQ 4 we have advocated the idea of an extensible link engine, which is essentially a stretching out of the navigation axis. Microcosm provides the generic link and filter concepts (see [Hill,Hall'94]) and DHM provides the notions of a location specifier and a reference specifier (see [Grønbaek,Trigg'96]).

Yet, we classify Microcosm more domain level tailorable than DHM, because Microcosm is more flexible. Extending the storage or presentation axis in Microcosm, requires the hypermedia system designer to tailor a viewer application to include calls to the Microcosm API and to perform some trivial installation procedure. With DHM a hypermedia designer would spend a comparable effort extending the viewer application, but installing it into the DHM framework is more difficult since it involves writing specialised subclasses.

System Level Tailorability

Especially in collaborative and distributed settings (see OHS-REQ 5, OHS-REQ 6) the need for system level tailorability becomes apparent. To implement the concurrency control required in such settings, one needs to incorporate special locking or notification strategies. DHM relies on database technology and object-oriented inheritance (see [GrønbaekEtAl.'94]) to incorporate special locking strategies[7].

Microcosm —although extended for a distributed setting (see [Hill,Hall'94])— does not support different collaborative settings because there is no easy way to wrap additional locking or notification behaviour around system modules in a uniform way. This is largely due to the componentware approach taken by Microcosm, where it is very easy to customise the behaviour of one single module, yet very difficult to extend all modules with similar behaviour. In DHM it is more easy to incorporate uniform locking or notification behaviour because there one can rely on object-oriented inheritance mechanisms.

[7] Hyperform ([Wiil,Legget'92] and [Wiil,Legget'93]) is an example of an open hypermedia system that took a similar —yet more flexible— approach by permitting dynamic subclassing of locking and notification classes.

The same argumentation can be repeated for other system uniform behaviour such as logging, caching, authority control and integrity control. In a framework approach this is relatively easy to do, because one can inherit such behaviour from a special class. In a componentware approach this is more difficult since there is no way to change the internal implementation of components uniformly.

Configuration Level Tailorability

DHM relies on object-oriented framework technology to support system reconfiguration. The hypermedia system designer has to provide some specialised classes and register them into the framework's session manager [Grønbaek,Malhotra'94] and the framework architecture ensures that they get called on the appropriate moment. Microcosm trusts on its extensible free-format message passing protocol to achieve reconfiguration [Davis,Knight,Hall'94]. The Microcosm link service passes a message to all appropriate modules which have to decide for themselves whether they want to react.

The difference we observe here is a difference between a centralised versus a decentralised approach. Frameworks in general, always have a tendency to centralise the system configuration making it easier to reuse a system configuration across different framework incarnations (see the discussion on object factories - p.54). Componentware in general has the tendency to distribute the system configuration over the participating components, making it easier to perform local reconfigurations. Both approaches have their advantages and disadvantages and the one approach can not be preferred over the other.

We conclude that the three levels of tailorability are appropriate for classifying open hypermedia systems. Also, these three levels of tailorability allow us to differentiate between the componentware approach (represented by Microcosm) and the framework approach (represented by DHM) for open hypermedia systems. The componentware approach is more domain level tailorable but less system level tailorable than the framework approach. As far as system level tailorability concerns, we see that both approaches are different and that there is no objective criterion to prefer one over another.

In fact the notions of domain level and system level tailorability reveal the inherent distinction between both approaches. Componentware is just about making it easy to tailor one component (i.e., one point on a design axis) so is always more domain level tailorable. On the other hand, the framework approach emphasises on reusing the same solution over a range of components (i.e., a complete design axis) so is more system level tailorable.

There is one concluding remark to make on the appropriateness of the three levels of tailorability. As follows from the above discussion, the three tailorability levels cover all the issues in the requirement list and thus the three central themes of open systems — interoperability, extensibility and distribution. Indeed, domain level and configuration level tailorability are primarily concerned with interoperability and extensibility, while system level tailorability covers aspects that have to do with distribution and sharing. *So we assert that the three levels of tailorability can be generalised to any kind of open system.*

A Framework Browser Scenario

Above we presented the Zypher design space and the three tailorability levels, which form the viewpoint that has been driving our experiments. To measure the strength of our viewpoint and to illustrate the scope of open hypermedia we envision, we pick up the idea of an open hypermedia usage scenario. The structure of our scenario comes from the sample published on the world-wide web at http://www.csdl.tamu.edu/ohs/; although we removed some subtitles we found less important and added one title "Our Results" explaining how far our experiments succeeded in achieving the goals of the scene.

Indeed, the scenario describes the vision that has been driving the design of the Zypher open hypermedia system and that has been partially realised in the many experiments we conducted. Nevertheless, the full completion of this vision has yet to be realised.

The scenario draws upon our experiences developing a Group Decision Support System (GDSS) [Kenis'95]. GDSS research is part of the larger computer supported co-operative work (CSCW) aiming to support the negotiation process in large groups and to improve the quality of group decisions. The emphasis of this research is on finding suitable communication structures that ameliorate the knowledge transfer between the participants, this way ensuring that more alternatives are considered with more wisdom. A computer may support diverse aspects of such a structured communication process, such as

- the identification of potential conflicts and possible compromises;
- the classification of different ideas, alternatives and opinions within the group;
- the shift from one decision making phase to another (support the 'chair' function);
- the publication of intermediate reports and final decisions (the 'secretary' function).

The scenario itself describes a framework browser as an integrated set of tools that supports an interdisciplinary team in the development of an object-oriented framework. This is another kind of engineering process than the Boeing counterpart presented in [Malcolm,Poltrock,Shuler'91]. We do not claim that this scenario represents the best approach to object-oriented software engineering, nor do we state that the scenario covers all aspects of the framework development process that should be supported by an imaginary framework browser. However, we are confident that the scenario represents a valuable approach towards software engineering and we use citations from [Goldberg,Rubin'95] to sustain our viewpoint.

The reader should be aware that object-oriented frameworks play several roles in this scenario. First, frameworks are the application domain for which we describe an imaginary open hypermedia system. In this sense, the term "framework" denotes all possible frameworks that a group of software engineers might conceive and it is represented by the GDSS framework in this scenario. Second, the scenario is about an integrated set of tools that support a framework development process. The integration is accomplished by means of an imaginary open hypermedia system with three levels of tailorability. In this sense, the term "framework (browser)" stands for tools and is represented by an imaginary framework browser in this scenario. Finally, the scenario illustrates the contributions of the Zypher experiments, which is a concrete incarnation of the above "framework browser" implemented using framework and open hypermedia technology. In this sense, the term "framework" denotes one particular framework providing a reusable design for the domain of open hypermedia frameworks and is named Zypher.

Especially when giving examples and discussing our experiments, it is hard to distinguish between those three roles, because in our experiments all roles are played by one and the same Zypher framework. Note that the Zypher framework has been implemented in the VisualWorks/Smalltalk environment and that the Zypher framework browser incorporates most of the programming tools that come with this environment, such as the system browser, class browser and method list browser. We refer to [Goldberg'84] for more information about these code browsers.

Scene 1: User Interface Prototype

Dirk is a sociologist and has been involved as a consultant in several projects supporting decision making in policy problems. Dirk applies a methodology based on the Delphi method comprising three cycles of paper questionnaires that are distributed among the group members by surface mail. Between each cycle, Dirk analyses the answers, rewrites the questionnaire and redistributes them among the group members. This process takes about a year to produce the final report and Dirk would like to speed it using computers. To illustrate his vision, Dirk uses HyperCard as a user interface prototyping tool. He authors cards for all the screens he believes are necessary in such a GDSS and links them together to create the effect of a working system.

Goal(s) of this Scene

"*Make the end user an insider. Successful teams enrol the end user as an insider, an active member of the product team.*" ([Goldberg,Rubin'95], p.294)

This scene stresses the importance of end user participation, already in the very early stages of software design. The scene shows that a framework development environment should support end users to prototype the user interface and control flow of their system. HyperCard represents the level of end user programming required in such an prototyping environment.

Character(s)

Dirk is the only character in this scene, playing the role of a requirements provider, aggressive tester and enthusiastic supporter ([Goldberg,Rubin'95], p.305). Dirk is a heavy user of all kinds of desktop publishing programs and is able to use such programs to express his ideas. He claims he does not know anything about writing programs, but he has experience with scripting languages and macro facilities that come with the desktop publishing software.

Framework Browser Requirements

To achieve true commitment, end users should play an active role and be able to build parts of the final program themselves. As end users typically do not have the technical expertise to program in full-fledged object-oriented software development environments, a componentware programming environment (like HyperCard) should be integrated with the framework browser. However, to achieve maximum effect, the components offered in the end user environment should correspond with concepts in the application domain. In our scenario, the prototyping environment should include components representing questions, answers and participants besides the generic ones like field, button, label, icon that come along with traditional componentware.

A framework browser should include a componentware programming environment with components tailored towards the target application domain (OHS-REQ 4: Data Models; Domain Level Tailorability).

Our Results

The Zypher experiments do not support the above requirement. However, we were actively engaged in the ApplFLab project, investigating how one could make user interface builders incrementally refinable ([SteyaertEtAl'94], [SteyaertEtAl'95]). This research starts from the assumption that user interface builders are essential tools for the development of modern applications, and argue that the state of the art of the field lacks a way of incorporating new user interface components. The ApplFLab environment can extend its set of user interface components using a special purpose meta-level interface. We plan to combine the ApplFLab environment with the Zypher framework browser in the future.

Scene 2: Analysis and Design

Having finished the GDSS prototype, Dirk starts to look for a group of people that can turn it into a working software system. He pays a visit to Patrick, a member of the computer science department of his university. Patrick happens to have a great deal of experience with object-oriented frameworks and he convinces Dirk to build a GDSS framework. Dirk fancies this idea, because he wants to try the future GDSS in both synchronous-proximate (i.e. all participants are working in the same room at the same time) and asynchronous-distributed (i.e. the participants are connected by a WAN-network and participate at different moments in time) settings and Patrick promises that he will be able to do both.

Patrick is well aware of potential pitfalls when building frameworks, certainly for a domain where he has no experience with. He decides that a "design by wandering around" method is probably the best way to cope with the uncertainties and tries a prototyping approach. To kick of, he starts to browse through the documentation Dirk provided, meets a few times to discuss GDSS ideas and gradually builds up an understanding of what Dirk had in his mind when building the user interface prototype. Patrick finally writes a set of analysis and design documents using his favourite tools (i.e. the Microsoft Word editor and a public domain OMT editor). Patrick uses an open hypermedia system to create numerous hypermedia links representing relationships between the OMT-diagrams and the analysis and design documents.

Goal(s) of this Scene

"*Tools. […] Much of the effort involved in analysis and design involves handling information with complex interdependencies that have to be created and maintained over long time periods. Tools must be available to handle the information management aspect of a (analysis and design) method.*" ([Goldberg,Rubin'95], p.346)

This scene illustrates that framework development is more than writing an object-oriented program and that a framework browser should support other related tasks as well. Moreover the scene emphasises that software engineers have their working habits and their favourite specialised tools, so that a framework browser must delegate specialised tasks to third party applications.

Character(s)

Patrick enters the scenario. Patrick is the framework designer ([Goldberg,Rubin'95], p.497), the person who determines the architecture for a generic set of parts that can be used to form GDSS applications. In this scene, Patrick designs an initial set of concrete parts to illustrate how the generic parts fit together to form a particular GDSS application.

Framework Browser Requirements

To support the wide range of tasks involved in framework development, the framework browser must be able to integrate any other application. However, the two example applications are quite different with respect to ease of integration. Microsoft Word is an example of a software package that incorporates interoperability standards (i.e. DDE and OLE) and is highly tailorable (i.e. the VisualBasic scripting language) and thus well suited for integration with the framework browser. The public domain OMT editor stands for all other applications lacking such facilities. With the growing support of operating systems, one can expect that more and more applications will adhere to interoperability standards, yet legacy applications will always be around. Techniques to deal with such legacy applications (i.e. like the "universal viewer" [Davis,Knight,Hall'94]) are essential.

A framework browser should be able to integrate any other application, irrespective of the degree of tailorability of such client application (OHS-REQ 3: Applications; Domain & Configuration Level Tailorability).

Our Results

We have integrated Microsoft Word as a third party document viewer. We used the macro facilities in Microsoft Word to accommodate for the necessary glue and implemented the inter application communication with DDE (Dynamic Document Exchange) under Windows.

The object-oriented implementation of the Zypher framework, makes it fairly easy to reuse the solution to communicate with other applications supporting DDE. Other interoperability standards (i.e. OLE, OpenDoc) could be incorporated if we want to.

However, the glue code in Microsoft Word is tedious to write, large in volume and hard to maintain. There is a definite need for some kind of reuse mechanism to support the incorporation of external application and this need is even stronger as far as communication with legacy applications concerns. We did not experiment with legacy applications due to a lack of resources. Standardisation efforts, and especially the idea of a dual protocol shim like described in [Davis,Lewis,Rizk'96], may answer the need for reusability.

Scene 3: Implementation

Patrick starts implementing a series of prototypes. As his knowledge of the GDSS domain grows, he sees that some of the assumptions in the analysis and design documents are incorrect and that others or missing. As is encouraged in incremental software development, he modifies the analysis and design to reflect his new insights.

To convince Dirk of the progress of the project, Patrick asks Koen to join the team and to develop a really attractive user interface based on the HyperCard prototype. The two of them form a real team now, working on the same artefact and need to synchronise their efforts. Patrick and Koen encounter two types of synchronisation problems.

The first problem is the maintenance of the relationships between evolving analysis, design and implementation. For instance, Patrick carefully designed that part of the GDSS framework that exchanges messages between participants, as this part must vary when building a GDSS-system for a synchronous-proximate or an asynchronous-distributed setting. Patrick included a link from the section in the design document to the places in the code of his prototype that implements this design. However, Koen has code in his prototype that implements the same functionality and Patrick wants the target of his link to include Koen's code as well. In fact, Patrick wants his link to include all current and future code that implements his design; he needs some kind of "smart" links that establish the navigation relationships dynamically based on specific knowledge about the framework architecture.

The second problem has to do with simultaneous access to shared data and what is called "tele-presence". Sometimes Patrick is editing the analysis and design documentation while Koen is reading it, so Koen and Patrick need to be aware that they are simultaneously working with the same document. The viewers on Patrick's and Koen's machines must show the document in a special colour to illustrate they work in a simultaneous access mode, and each time Patrick saves the document Koen must see the latest version. Of course, this schema must work with all applications used to view and edit parts of the framework.

Goal(s) of this Scene

"*Framework Goal. Set up a development environment for all team members that enables the team to create, maintain and deliver applications.*" ([Goldberg,Rubin'95], p.376)

This scene portrays that incremental development and teamwork implies facilities for consistency maintenance and integrity control. Professional software development environments may include such facilities, but when a framework browser incorporates third party applications it cannot rely on these facilities.

Character(s)

Koen comes on stage now, playing the role of an implementor in an application production team ([Goldberg,Rubin'95], p.282), specialised in user interface design. Koen stands for other characters with other specialisation's (i.e. databases, networks, statistics) as well. The point we want to make is that framework development is teamwork, so a framework browser must support working in teams.

Framework Browser Requirements

Traditional hypermedia systems use static point-to-point links to structure navigation relationships, which means that the author of a hypermedia system must maintain the connections between the sources and targets of the links manually. However, in environments where those relationships change often, static links become a maintenance nightmare. One way to deal with dynamic structures is to have some kind of smart links that use knowledge about the information to infer relationships. In the scene, Patrick would create a 'smart link' that uses internal data structures of the implementation environment to return all occurrences of the message implementing the design.

> A framework browser should be able to incorporate smart links incorporating knowledge from inside the implementation environment (OHS-REQ 4: Data Models; Domain Level Tailorability).

To implement tele-presence, the system must ensure integrity between different viewers, simultaneously manipulating the same data. This implies that the framework browser must be able to monitor open, change and close events and pass them to other viewers. Those viewers are in principle unaware of each other and may or may not run on the same machine.

> A framework browser should be able to trap important events and notify other viewer applications, irrespective of the type or number of viewers involved (OHS-REQ 6: Users; System Level Tailorability).

Our Results

Zypher is equipped with a fully extensible link engine (see "resolver" - p.94) and we have plugged in algorithms for 'smart links' between analysis and design documentation on the one hand and the implementation on the other hand. Analysis and design documents may include HTML style anchors[8] with a special "browse:" keyword. This keyword identifies a smart link referring to an element of the implementation. Possible references are "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).

Zypher does not include any facilities for concurrent users, so we could not yet demonstrate tele-presence. However, using the meta-object protocol we can trap all open, close and change events (see "meta-objects" - p.116) and we implemented a layered event passing mechanism on top of it (see "events" - p.107). Using this mechanism, we have been able to synchronise two home cooked viewers (i.e. HTML and text) that are completely unaware of each other. When we save a HTML representation in a certain text file using the simple text viewer, the HTML viewer opened on the same text file updates its contents automatically. The same scheme could have been used to synchronise the contents of two third party applications running on the same machine (i.e. the Notepad and Microsoft Word) viewing the same file. Synchronising viewer applications running on different machines is more difficult but would be feasible as well: besides the obvious need for a high level communication

8 In the Zypher experiments, we use embedded links to avoid the editing problem (see OHS-REQ 1).

channel between the two machines, it would involve an extension of the meta-object protocol to pass events over distributed machines.

Scene 4: Reuse Library

After a few analysis/design/implementation iterations, Patrick and Koen succeed in building two GDSS's; one for a synchronous-proximate setting and one for an asynchronous-distributed setting. Dirk is now testing the software with a number of groups and is quite pleased with how it works.

Patrick and Koen actually build a reusable design for the GDSS domain. However, there solution probably contains parts that may be reused in other CSCW frameworks. They hand over all the analysis, design and implementation to Roel, who is responsible for the identification of reusable assets and for storing them in the computer science department reuse library. To maximise access, this library is accessible via the world-wide web.

Roel studies the received material and via discussions with Patrick and Koen he identifies those parts of the analysis, design and implementation that may be reused in other frameworks. For each reusable asset, he writes a design pattern, including an analysis section (based on some ideas in the original analysis), a contract section (based on the original design) and an implementation section (including references to the original implementation).

Roel is now ready to store the new design patterns in the library. As far as the design pattern document concerns, this is not so difficult: he relies on the multitude of HTML converters available. The references to the implementation impose a special problem however. The tools supplied with typical programming environments provide a narrow, class-oriented perspectives on class- and object structures. Design patterns are usually spread over several classes and only a few methods matter to understand one particular design pattern. Roel concludes that the tools provided with the programming environment are not suited as viewers for the implementation of design patterns. Roel decides to build special purpose viewers himself, using the tools provided by the open hypermedia system (i.e. a set of components to assemble design pattern views). Moreover, he wants to translate those views into HTML forms, so that the reuse library would accessible through the web.

The design pattern library gets larger and larger, and Roel encounters consistency maintenance problems. A design pattern does not work alone, but is supposed to be used in combination with others. However, it is quite difficult to manage links between design patterns that are known to be working together. Roel decides to use the extensible link engine of the open hypermedia system to plug in special algorithms that help in managing the consistency of the library. The idea is that if design patterns work together, the implementations of both design patterns will overlap to some extent. Roel extends the hypermedia system so that when a new design pattern is added, its contents is scanned for all references to the implementation and those references are stored inside a special database. This database is used to infer relationships between design patterns that use the same implementation.

Goal(s) of this Scene

"*Framework Goal. Set up a structure in which to plan and manage the process of acquiring, distributing, and maintaining reusable assets throughout the organisation.*" ([Goldberg,Rubin'95], p.223)

This scene illustrates the different approaches towards reuse within one framework and reuse across frameworks. To accomplish the latter, some kind of reuse library must be set up. There again —certainly with the advent of the world-wide web— an open hypermedia system can play an important role to make the library easy accessible.

Character(s)

Roel is the so-called reuse librarian ([Goldberg,Rubin'95], p.500), responsible for certifying, classifying, and storing new reusable assets in the corporate or project library. Much of the reuse assets in the library are based on the design pattern catalogue published in [GammaEtAl'93] although Roel collected many domain specific design patterns as well.

Framework Browser Requirements

Being able to call upon services of third party applications to view information is one of the selling arguments for an open hypermedia system. Yet, third party viewer applications are not always able to show the desired information, partly because this information is stored within the link structures managed by the hypermedia system. So an open hypermedia system must provide facilities that allow the hypermedia author to build its own viewer applications easily, probably based on some kind of componentware approach.

> A framework browser needs powerful, easy-to-use programmable tools for building specialised viewer applications (OHS-REQ 2: Data Formats, OHS-REQ 4: Data Models; Domain & Configuration Level Tailorability).

The world-wide web is reaching the status of a universal exchange medium. An open hypermedia system should make it easy to exchange information with the world-wide web.

> A framework browser must be able to exchange information over the world-wide web (OHS-REQ 2: Data Formats; Domain & Configuration Level Tailorability).

To support a library function in a framework environment, an open hypermedia system needs tools to link a new library item with the items already in the library.

> A framework browser should be able to incorporate smart links to support consistency maintenance in large reuse libraries (OHS-REQ 1: Size, OHS-REQ 4: Data Models; Domain Level Tailorability).

Our Results

Zypher comes with a library of components that can be assembled programmatically to form new code browsers. We plan to integrate these components with the ApplFLab environment to attain a componentware approach (i.e. similar to [Wuyts'95]). Using VisualWave (a Smalltalk tool for generating HTML forms out of Smalltalk window specifications) we can publish these code browsers on the web quite easily.

To support the consistency maintenance of the design pattern library, we have extended Zypher's link engine to incorporate knowledge about design patterns. When the librarian stores a new design pattern in the library, he must provide a list of all the messages participating in the corresponding framework contract. All these messages are stored in a special database and Zypher's link engine queries this database to generate a pop-up menu of all related patterns.

Conclusion

We have presented our perspective on open hypermedia, which forwards tailorability as an distinctive property of open hypermedia systems, encompassing both the style and adaptability aspects of open hypermedia. We have identified three levels of tailorability and have defined them based on the notion of a design space.

A design space represents a range of systems in an n-dimensional axes system, where each axis represents a fundamental characteristic present in all systems but differs depending on the exact system one is considering. We propose three axes to model a hypermedia design space: a storage axis enumerating all possible information repositories, a presentation axis enumerating all possible viewer applications and a navigation axis enumerating all techniques to specify navigation relationships.

The three levels of tailorability are defined based on the adaptability of the axes system. We identify domain level tailorability which corresponds with the addition of extra points to an

axis; system level tailorability, which corresponds with wrapping uniform behaviour around points on one axis; and configuration level tailorability, which corresponds with changing the relationship (in the mathematical sense of the word) between points on the design space axis. The puppet-master metaphor is presented as a memory aid.
An important property of these three levels of tailorability is that they can be generalised for any kind of open system. This follows from the observation that the three levels of tailorability cover the important themes in open systems: interoperability, extensibility and distribution.

We have shown that the proposed perspective is appropriate for classifying open hypermedia systems by contrasting two representative systems Microcosm (representing the componentware approach) and DHM (representing the framework approach). We have concluded that the componentware approach is more domain level tailorable but less system tailorable. As far as configuration level tailorability concerns, we have observed a difference but we are unable to prefer one approach over the other.

We have illustrated the value of the proposed perspective with a scenario of a framework browser. A framework browser is defined as an integrated set of tools that support the development of object-oriented frameworks, and we have used the framework browser scenario to relate the three levels of tailorability with the requirement list part of the open hypermedia definition. This scenario describes the vision that has been driving the design of the Zypher open hypermedia system and that has been partially realised in the many experiments we conducted. Nevertheless, the full completion of this vision has yet to be realised.