Vrije Universiteit Brussel
Faculteit Wetenschappen

Virtual Hypertext based on Paths and Warm Links

Serge Demeyer

Techreport vub-prog-tr-94-10

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

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

Virtual Hypertext based on Paths and Warm Links

Serge Demeyer
Brussels Free University / Faculty of Sciences / PROG
Pleinlaan 2
B-1050 Brussels, Belgium
fax: (32) 2 641 34 95 (629 replaces 641 from June '94 on)
e-mail: sademeye@vnet3.vub.ac.be

ABSTRACT

Throughout the last years a huge amount of work has been devoted to the definition of hypertext models. Even more resources have been directed towards the domain of virtual (dynamic/ computational) hypertext, among others motivated by the idea of building open systems. Surprisingly enough, almost nobody stressed the role of the underlying model in such virtual systems.

That is precisely the aim of this text: to define a general hypertext model that is able to support the notion of virtuality. Our assertion is that the combination of the ancient concepts 'Paths' and 'Warm Links' provide just the extra support needed. Moreover this allows for a model where links are but one of the possible ways to relate nodes.

While experimenting with the model, an interesting question arose: do bi-directional links fit into a virtual model ? This paper attempts to answer the question.

We chose a constructive approach, because our aim was to create a laboratory where ideas concerning virtual hypertext might be explored. We applied recent viewpoints from the field of software engineering (namely object oriented frameworks and mixins) to assist the iterative design process.

In order to show the value of the work, we have implemented two prototype applications. The first is a browser for viewing (Smalltalk) source code which includes query facilities, the second is an electronic agenda. These experiments demonstrate three desired properties of the model: the applicability (considering the differences between the prototypes), the open endedness (since it is able to establish hypertext structures on top of underlying foreign constructions) and the extensibility (while building the applications, we continued to expand the model).

KEYWORDS: Virtual Hypertext, Open Hypertext, Extensible Architecture for Hypertext, Path, Warm Links

1 INTRODUCTION AND INSPIRATION

In this part we will give an overview of the most important streams of research that influenced this work. As has been stated in the abstract, the main idea is to define a hypertext model that supports the notion of virtual hypertext. The first sections are devoted to the subfields of the hypertext research dealing with these topics: section 1.1 outlines the different models proposed in the hypertext literature; section 1.2 sketches the virtual hypertext domain. The third section presents some recent insights from the field of object oriented software engineering.

1.1 Hypertext Models

During the last years, many attempted to define a hypertext model. Motivating factors for such experiments were: comparing systems, coining terminology and exploring information exchange facilities. Sometimes designers applied general purpose tools and methodologies, sometimes no special formalism was used; some modelled general system architectures, others focused on special features.

HAM [HAM] and Neptune [NEP] introduced a layered approach (user-interface level/ abstract machine level/ database level) and the notion of contexts to partition data. Garg [GAR] defined a formal model that was able to impose abstraction structures (aggregations, generalisations, revisions, ...) on the information network. Trellis [TRE] was built upon the well-known formal computation model of petri-nets. Lange [LAN] started from a general purpose data-modelling technique (VDM) to define a set of abstract data types. The Dexter-model [DEX] combined a layered approach (run-time layer/ storage layer/ within-component layer) with a formal specification language (Z) to define a commonly accepted model (see [CA'94]). The GOOD-model [GOO] applied a general purpose database model to define the basic node-link model. Afrati and Koutras [AFR] extended the work of Garg with anchors, scripts (for presentation specific computations), and queries. Beeri and Kornatzky [BEE] augmented a basic node-link model with a structural query mechanism. Lucarella [LUC] used inference networks (nodes are facts, links are weighted rules) with fuzzy logic. Schütt and Streitz [SCHU] combined the HAM and Dexter approaches into database model for a relational database. ABC [ABC] did a similar job for distributed (data)servers. The "Nested Context Model" [CAS] paid special attention to the interface between the storage-layer and the runtime layer. HyperSet [PAR] employed set-theory to argue that links are not necessary for navigation. HDM [HDM] was tailored for large engineering design tasks. MacWeb [MWB] attached type information to nodes and links, and methods on types. As such, the model was able to perform various computations (among others tuning the behaviour of the system). Multicard [MUL] viewed links as communication channels between processes manipulating the hypertext contents. De Bra et al. [BRA] augmented the node-link-anchor model with constructors like Composition, Tower and City, which turn it into an extensible data model. HYDESIGN [HYD] is a model with a rigid hierarchical backbone but provides aliasing nodes for overlapping hierarchies. Several higher order nodes and link types are provided, which gives rise to flexible building blocks.

Obviously there is no need for "yet another hypertext model", especially if one regards the fact that introducing new hypertext models goes against the integration idea that everyone agrees will be the key success factor for the hypertext community.
We chose to make another attempt, however: the motivation being that we needed a laboratory-set-up to experiment with different approaches in the virtual hypertext field.

1.2 Virtual Hypertext

An idea that has received even more attention throughout the years is the notion of virtual (dynamic, computational) hypertext. Especially since Halasz [HAL1] identified this as one of the issues for the next generation of hypertext systems[1]. We quote: "In the current model, nodes and links are extensionally defined, that is nodes and links are defined by specifying the exact identity of their components. In contrast, virtual structures are defined intentionally, that is by specifying a description of their components. The exact subcomponents of a virtual structure are determined by a search procedure whenever the structure is accessed or instantiated."

This vision has been explored in several experiments from different angles.
Very soon, people felt that the notion of links was too static and coupled links with computations: activating a link computed the next point to visit. We encountered this property in early systems like KMS [KMS], Guide [GUI] and HyperCard [HPCD]. Today, computational links are almost standard: they are included in the Dexter reference model [DEX], and as such found their way in systems like MicroCOSM [MICR], MacWeb [MWB], WEBS [WEB], DHM [DHM], Hyperform [HFR].
Trellis [TRE] is a special case. Links are inherited from an underlying computational model (petri nets). One of the experiments was to adapt the hypertext structure based on the behaviour of users while browsing [TRE-HT'91].
Combining searching and information retrieval techniques with the notion of links provided another line of experiments. Links as the triggers for search procedures appeared in the medical handbook [MED1, MED 2], SuperBook [SUP], Document Examiner [DOC], Intermedia [COOM] and many, many more. Some tried to use the structure of the hypertext network to guide searching ([CRO], [FREI]) or to change the search strategies depending on the users response ([MED2, WALT]).
Volatile hypertexts [VOL] have been tackled by discarding the notion of links completely. SaTellite [PINT] uses an affinity browser where nodes are displayed in a two-dimensional representation and the distance between them gives a clue about their affinity. HyperSet [PAR] classifies information into structured sets. Aquanet [AQU] tries to deduce implicit relations from spatial layouts.
The idea of virtual nodes took longer to find its way into the literature. An early experiment was conducted in [SCH] where a simulation model was manipulated. PhiDIAS [PHI] used virtual nodes to provide automatic restructuring of information depending on user input. Bieber [BIEB] describes a Hypertext shell for a decision support system. The World Wide Web [WWW] hypertext system has virtual nodes that link the system with the contents of various news servers, ftp-sites, ... Nanard & Nanard [MWB - HT'93 p. 51] report on synthesised nodes: nodes constructed on the fly by copying parts of other nodes.

Virtual hypertext seems to be a domain rich of ideas and currently various hybrid forms of the above approaches are studied. Nevertheless, none of the proposed hypertext models (see section 1.2) provide constructs to support computation: it is seen as "an orthogonal dimension of all modelling constructs" [BRA]. We agree that orthogonality is an important factor in design, yet we believe that the notion of computation needs support from the other axes. To be more precise, computations need communication channels (to store and retrieve data) and contexts (to connect these channels and to hold status information).

[1] Actually Halasz reviewed the issue in [HAL2]. Virtual structures as such is not on the research agenda anymore; it is replaced by "Ending the tyranny of the Link" and "Open systems".

1.3 Object Oriented Frameworks

Applying techniques from the software engineering world (especially object orientation) is not new to the hypertext community. Intermedia advocated the use of object oriented languages [MEYR] and later object oriented databases [SMI] in building the system. Lange [LAN], Schütz & Streitz [SCHU], WEBS [WEB], ABC [ABC (HT'93)] moulded their model in a class hierarchy; DHM [DHM] and HYDESIGN [HYD] implemented their systems on top of an object oriented database. Recently, some aimed at building extensible systems: De Bra et al. [BRA] exploit generic modelling constructs; Hyperform [HFR] benefits from an extension language based on Scheme.
Our goal was to implement a laboratory-set-up for experimenting with ideas concerning virtual hypertext. In such a setting a "design by wandering around" approach is most appropriate. Extensibility is a key factor as our model should be able to grow with emerging needs. To assist in this dynamic process we adopted recent techniques from the field of software engineering. It is beyond the scope of this paper to give an overview of this field, but the following paragraph will summarise the ideas we find important.

It is generally accepted that "Object-Oriented programming permits the reuse of design as well as programs" [JO/RU]. This capability helps a lot in an iterative design process since it reduces the turn-around time in-between the iterations. However, "software reuse does not happen by accident, even with object oriented languages" [JO/FO]. Object oriented frameworks are constructs that enable software reuse in an object oriented language. "A framework is a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuse at a larger granularity than classes" [JO/FO]. As frameworks themselves can be refined [JO/FO, HELM], they typically impose a layered structure on the design.

A true engineering discipline requires a set of tools to solve routine design problems quickly and reliably [JO/RU]. In (software) engineering practice, such tools are called design patterns. We did benefit a lot from the catalogue of object-oriented design patterns described in [GAM, PATT]. An additional bonus was obtained by using mixins to allocate the implementation of design choices. "A mixin is an abstract subclass; i.e. a subclass definition that may be applied to different superclasses to create a related family of modified classes" [MIX1]. Mixins are helpful since they allow to reuse implementations. For a discussion on mixins see [MIX2].

2 THE MODEL

The model itself is a variation on the traditional link/node model (links are not an essential part of the model !) found in most hypertext systems [CON]. It is extended with the notion of 'anchors' [INTERMEDIA, DEX], 'warm links' [INTERMEDIA], 'paths' [ZEL] and -in a second layer- 'contexts' [HAM] and 'links'. The layering of the model enables us to regard links as not essential (links are but one of the possible ways to interconnect nodes).
Sections 2.1 and 2.2 list the operations defined in the first layer. Section 2.3 explains the second layer of the framework. Section 2.4 reports on how we brought virtuality in the model. Section 2.5 discusses the conflict between bi-directional and virtual links.

Since we have deliberately chosen for an object oriented data model, we use classes and messages as basic building blocks for our framework. The first layer of the framework (sections 2.1 and 2.2) will be listed explicitly.

All the classes below are abstract classes [JO/RU, JO/FO]. As such, none of them contain variables (although subclasses might use variables to implement the protocol).

2.1 The Basics

Like in most hypertext models, we state that every hypertext object should have a set of named attributes associated with it (typical attributes might be: creation time, original author, last modification time, last modification author, global identifier, ...).

```
Object subclass: HypertextObject

attributeAt: Name -> AttributeObject
{Return the value of the attribute with the
given name.}

hasAttributeAt: Name -> Boolean
{Answers whether the given name is used as an
attribute-name for the receiver.}
```

We want attribute values to be of type AttributeObject, not listed here for reasons of brevity. There is no method for setting an attribute to a given value because there might exist objects that are not able to hold such attributes (the attributes might be virtual). Other overall functionality (printing, displaying, saving to disk, saving to database, …) is not part of the model but should be considered in an implementation.

2.2 Anchor, Node and Path

Nodes hold some piece of information (the contents) and are connected to other nodes to form an information network. The points where the network connects to nodes are called 'anchors'. An anchor may be any selection in the contents of the node. (see [INTERMEDIA] for a discussion on anchors). A hypertext reader will navigate through the information space from one node to another by activating an anchor on a node which will trigger the corresponding link and lead the reader into another place (another anchor in some other node). The result of this activities forms a path through the hypertext.

We begin by introducing the classes Anchor and AnchorNodePair. The latter is needed for technical reasons:

when following a link the hypertext reader arrives in a place identified by an anchor and a node so the model needs a way to combine them in one object. The protocol for the former is rather small since we do not want to restrict the possible anchor types.

```
HypertextObject subclass: AnchorNodePair

anchor: -> Anchor
{Returns the anchor of the aggregation.}

node: -> Node
{Returns the node of the aggregation.}
```

In the following definition of an anchor we meet a typical example of a factory method [PATT]. The @ operator returns an object, subtype of AnchorNodePair. The decision on the returned values exact class is delegated to subclasses (since AnchorNodePair is an abstract class, it is not supposed to have instances).

```
HypertextObject subclass: Anchor

@: Node -> AnchorNodePair
{Combines the receiver with the given node into
an aggregation.}

value: -> Object
{Answer some kind of value that may be used to
identify this anchor.}
```

Many researchers agree that links in hypertext systems should be considered less important. That is the main reason why we postpone the introduction of links (together with contexts) to a later stage. We believe that the idea of connecting nodes is crucial to hypertext and we choose to focus on anchors instead of links. Activating an anchor on a node follows the connection at the given anchor, and answers the anchor and node on the other side. An extra parameter (type Path) is passed, and will be used to resolve the link.

Objects that are used as contents of a node should obey the protocol defined in ContentsObject. This class is not listed here due to space considerations. It might contain implementation specific properties (printing, displaying, persistence, ...).

```
HypertextObject subclass: Node

followAtAnchor: Anchor * Path -> AnchorNodePair
{Returns the node and anchor-pair at the other
side of the given anchor. Use the path to store
the new status of the navigation.}

validAnchor: Anchor * Path -> Boolean
{Answer whether a given anchor is valid for this
node and path.}

validAnchorsDo: Computation * Path -> Nil
{Enumerate all valid anchors (validation using
path) and apply the given computation on them.}

contents -> ContentsObject
{Returns the contents of a node.}
```

The main rationale for this work was the ability to model virtual hypertext where the structure and contents of the network are computed at traverse-time. For computation it is essential to pass data around. This is accomplished through the idea of warm linking introduced in [INTERMEDIA]: "Warm linking allows a user to not only traverse a link from a selection in one document to a selection in another document but also to transport data across the same link". This idea of links as communication channels is also explored in Multicard [MUL]. (Note that we avoided to include links in the model by careful usage of the anchor and path concept.)

A node may request for information by pulling a certain connection (i.e. anchor). This will result in the execution of a pulled method with the node and anchor on the other side of the connection, which will compute the desired value. Complementary to these are the push and pushed messages, used for sending new values to other nodes. (Note that passing parameters is possible by using attributes of the path.)

```
Class Node (continued)

pullAtAnchor: Anchor * Path -> Object
{Request some data through a given connection
(identified by anchor) using the given path.}

pulledAtAnchor: Anchor * Path -> Object
{Some node requested some data through a given
connection (identified by anchor). Answer it
using the given path.}

pushAtAnchor: Anchor * Object * Path -> Nil
{Disclose some data through a given connection
(identified by anchor) using the given path.}

pushedAtAnchor: Anchor * Object * Path -> Nil
{Some node disclosed some data through a given
connection (identified by anchor). Receive it
using the given path.}
```

The only missing concept in our framework is the Path. In a minimal setting, paths can be regarded as an ordered collection of the nodes and anchors the hypertext reader has already seen.

Paths play a more important role in our model: every anchor operation will eventually be passed to the path, even the operations for validating an anchor. This means that paths can (and will) determine the navigation potential of the hypertext network. Another particular function the path may fulfil is the 'Presentation Specifications' layer in the Dexter reference model [DEX].

Implementors are able to plug in different behaviour by accommodating special paths. In section 2.3 we will use this ability to plug in 'links', section 3.2 will report on an implementation based on searching.

We assumed that the anchors that may be activated from a certain node depend more on the path than on the node, so we delegate the navigating functionality to paths (we will see that the model is open enough to explore other strategies when discussing chooser nodes in section 2.4). Thus the default behaviour for anchor operations on nodes is to pass them on to the path involved. Nodes can (and will) override this behaviour.
Anchor methods on nodes are examples of template operations [JO/RU]: an abstract algorithm defined in terms of one or more abstract operations. An abstract operation is not implemented by the abstract class but is left to subclasses to define. The anchor methods on paths are examples of abstract operations.

```
HypertextObject subclass: Path

isValidFrom: Node * Anchor -> Boolean
{Tell whether the given anchor may be triggered
on the given node.}

validFrom: Node * Computation -> Nil
{Enumerate all valid anchors on the given node
and apply the given computation on them.}

followFrom: Node * Anchor -> AnchorNodePair
{Activate the given anchor on the given node and
answer the anchor and node on the other side}

pullFrom: Node * Anchor -> Object
{Pull the connection starting in the given
anchor on the given node and answer the received
data}

pushFrom: Node * Anchor * Object -> Nil
{Push the given data through the connection
starting in the given anchor on the given node.}
```

2.3 Context, Link and PathWithContext

For reasons of brevity, and as this part of the framework is less relevant, we will not list the definitions of the classes and methods anymore.

Given a hypertext created with objects obeying the protocols in the previous section, one could easily implement navigation tools like we find in all hypertext systems. To create such hypertexts we need to implement the model, especially the way a path resolves the connections.
We have several possibilities here: set-based structures like in HyperSet [PAR], indexed search facilities like in QRL [QRL], implicit relations like in Aquanet [AQU], and many more. In fact, the model is open enough to experiment with different techniques, and in section 3.2 we will present such an experiment based on searching.

There is one option we (explicitly) excluded in the above enumeration: links. Indeed, many people agree that "Linking should be considered harmful" [YOUNG], and advocate that the hypertext community should "end the tyranny of the link" [HAL2]. We decided to retain links, but chose to isolate them in a separate layer of the framework. This way links are but one of the options in relating nodes. In what follows we will focus on this layer.

An issue that has been widely debated is the "lost in hyper space" phenomenon [CON]. One way to approach this problem is to reduce the possible links that may be active at a certain time. Both Intermedia [INTERMEDIA] and Neptune [NEP, HAM] followed this approach: the former with the concept of webs, the latter with the introduction of contexts. In both approaches links are collected in special structures (called webs or contexts respectively); only those links belonging to the current web (context) may be followed.
One of the nicer things of the Neptune approach is that contexts themselves are gathered in a hierarchy. Moreover Casanova et al. [CAS] defined formal support for (nested) contexts. That is why we chose context as the name for the component.

We define a context as a collection of nodes (nodes may belong to more than one context) and a collection of links. The links in the context should start from nodes part of the same context but may arrive in another context[2]. Various methods for manipulating nodes and links in a context are available.

A link is a unidirectional connection between (anchors of) two nodes each one attached to a certain context. The two sides of the link are designated departure and arrival implying the direction of the link. Accessor functions for anchors, nodes and contexts on both sides are defined in the link class.
Actually we are not completely satisfied with this definition of a link. With others [INTERMEDIA, NIELS] we agree that links should be bi-directional with multiple endpoints. We have tried to fit such a link in our model but faced some difficulties. See section 2.4 for a discussion on this matter.

Now we are able to build on the path concept. PathWithContext is defined as a subclass of Path with extra accessor functions for a current context, node and anchor. When a path activates an anchor on a certain node the link will be looked up in the current context and used to jump to (push, pull) the arrival side. The internal state of the path will be updated accordingly: pushing or pulling links will update the current anchor; following links will modify the current anchor, node and context.

[2] In the nested context model [CAS] there should exist a context in the hierarchy that is parent of the linked contexts.

2.4 Virtual Nodes & Links

As we have said in the introduction, computations need communication channels (to pass data) and contexts (to connect these channels and to store parameters). The former is provided by warm links (i.e. the push and pull operations on nodes), the latter by paths.

The proof of the pudding is in the eating: we have defined all the necessary components of the system, now it is time to come back to the notion of virtuality.
Integrating virtual nodes in the system was quite easy. At creation time a computation must be supplied to the virtual node. When calling the contents method the computation is evaluated and the result is returned. For virtual links an similar scheme was followed, the only difference being that the computation returns an aggregation object (i.e. a combination of an anchor, node and context).
We had several options for the parameters to be passed on the computations: we adopted a minimal solution by passing the virtual object and the path. It is essential to pass the path to the computation to enable link communication (pushing & pulling data). Passing the virtual object itself is the easiest way to deal with computations independent of the actual object they are representing. Attributes (section 2.1) in the path may be used to pass additional parameters.
An extra facility built into the virtual objects was the buffering of results to avoid recomputing the same value over and over again.
The choice of Objectworks\Smalltalk as the implementation vehicle has proven to be worthwhile. As computations (BlockClosure) are first class citizens it was quite comfortable to experiment with different parameter passing strategies, eager and lazy evaluation, etc. Nevertheless, once this experimentation phase is over, we believe that porting the framework to other Smalltalk environments or other languages (CLOS, C++) will not cause enormous problems.

An extra challenge was the notion of a virtual chooser node. In the applications we envisioned as our first testbed (see section 3.1) users would encounter several forking points: places where a range of options must be displayed and one must be chosen. We extended the model with the notion of a chooser node, having as contents a collection of (displayable) anchors. A chooser node has its private set of links, and thus overrides the default behaviour of nodes (passing anchor operations to a path). Integrating this concept into the framework was almost effortless. Afterwards we assembled a virtual chooser node from a chooser node, a virtual node and a virtual link.

2.5 Bi-directional versus Computational Links

With other designers [INTERMEDIA, NIELS] we feel that bi-directional links have nicer properties. However computational links prevented us from building this feature into the model.
Indeed, an intrinsic characteristic for computational links is that the arrival part is unknown until the link is computed. As a consequence it is impossible to follow the link in reverse mode before it is triggered (given the fact that there are no restrictions on the computation).
The solution might be twofold. The first is to stick with unidirectional links, but provide aggregation facilities to combine two (or even more) unidirectional links into one bi-(multi-)directional link. The other is to limit the computation power so links become reversible.
In the current implementation we use Smalltalk [SMALL] as the computation engine. This comfortable situation gives us the opportunity to experiment with various computation models (unification languages [UNI] in particular seem quite interesting) and this is one of the future directions we consider.

3 PROTOTYPE I: A SMALLTALK BROWSER

To illustrate the merits of the framework we will present a small prototype application implemented on top of our hypertext model. It consists of a class browser (demonstrating the use of virtual nodes and links) with a query mechanism (to reveal the possibilities of navigation by query instead of traditional link chasing).

We present the development as a process where the improvement of the application gives feedback to the design of the framework, this way showing the extensibility of our model.

3.1 The Class Browser

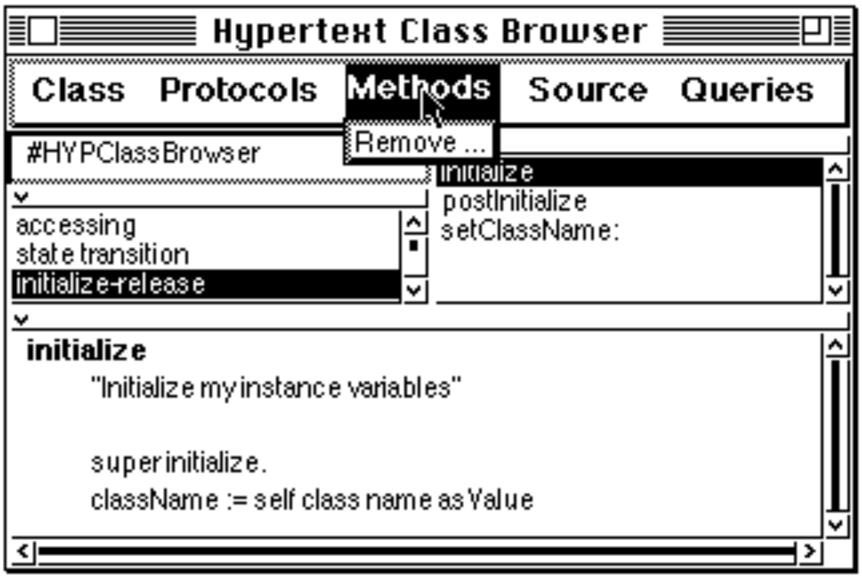

Figure 1: a view of the class browser implemented on top of the virtual hypertext model

The class browser is one of the programming tools available in a Smalltalk programming environment [SMALL]. Via the browser one can edit the class definition (instance & class variables, superclass, ...), the class comment (documentation) and the methods associated with that class. In the Objectworks environment, methods are grouped into protocols.

Figure 1 shows a screendump of our browser. A row of pull down menus are located just below the window title bar. Underneath one can find a text field containing the name of the class, two selectable lists (on the right a list of protocols, on the left a list of method selectors) and finally a small text editor. Selecting an item in the protocol list changes the contents of the method selector list (all method selectors in this protocol). Selecting an item in the method selector list displays the source for this particular method in the text editor. Users can edit this source. By choosing 'Accept' from the 'Source'-menu the method is compiled into the Smalltalk system. When no method is selected, the editor contains a template for entering new methods. The class definition is displayed when there is no protocol selected. After selecting 'Comment' or 'Definition' the corresponding class attributes are displayed in the editor. The Protocols and Methods menu are used to add/remove items to/from the respective lists. We will come back on the Queries menu when we discuss query based navigation in section 3.2.

Implementing this browser on top of a traditional node-link hypertext is not so difficult. One creates text nodes for the class definition, the class comment and every method defined on the class. A few chooser nodes are needed to link everything together (one chooser node for the protocols and a chooser node for every protocol in the list). All that is left to do is glue the interface actions to the appropriate link activations.
This can be done in a straightforward way but requires lots of glue code. We used the facilities of the model whenever possible. When appropriate, we extended the model to minimise the programming effort.
The first thing we did was pushing the state information the browser needs into the hypertext model. Indeed, when choosing 'Accept' from the source-menu the appropriate actions vary on the current mode of the browser: accepting a modified method requires other steps than accepting a class definition. This is normally implemented through a state variable that contains the current text mode. We supplied every text node with an extra link that connects it to a 'status' node. When the user arrives in such a node we push the node name through this link so that the status node is able to store the current text mode. We extended the model with a special node type to store pushed values.
Then we observed that a lot of code went into the translation of a single user action in a sequence of link activations. Again we extended the model, but now with the notion of scripted paths. A script is a sequence of <anchors, action> pairs, where an action stands for 'follow', 'pull' or 'push'. A scripted path is a path with a set of (named) scripts. Implementors are able to plug in code that is executed on every script transition, so certain interface effects may be hooked into the hypertext.

But the most important perception was that adding or removing methods and protocols was very hard. Besides the fact that the underlying Smalltalk system itself must be updated (this part of the job is inevitable), one needs to do all the bookkeeping tasks on the hypertext network itself (i.e. to add a method or a protocol one extra node and three extra links are necessary). This requires careful coding and a lot of knowledge about the hypertext structure inside the browser which goes against good software engineering

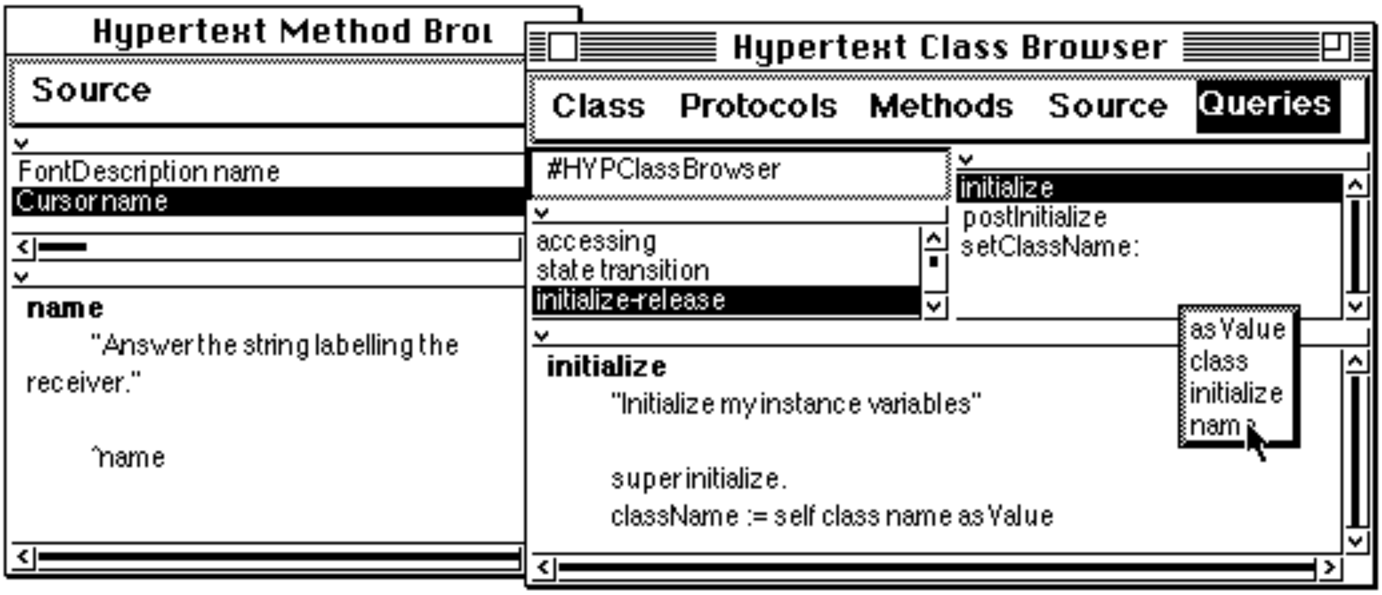

Figure 2: a view on selecting 'Messages ...' from the Queries menu

practice.
With the introduction of virtual nodes and virtual links the problem was solved. Modifying the hypertext graph has become obsolete. When adding or removing objects, the browser must update the underlying Smalltalk system. Pushing certain nodes forces the necessary recomputations and updates the display.

This experiment showed us that the model may be used to incorporate traditional static hypertexts. Adding virtuality enables an 'open' system: the application is then smoothly integrated with foreign data structures.

3.2 Querying the Smalltalk system

One of the appealing things of the Smalltalk browser is that it may be used to query the system. A user may browse all other implementations of a method, all senders of a message, all messages a particular method sends, all methods that access a certain instance variable, ... In our browser these queries are issued from the 'Queries' menu. Figure 2 shows what happens when a user selects 'Messages ...' from the 'Queries' menu. First of all a pop-up menu with all the messages in the selected method is activated; choosing one method opens a method browser (the window in the back).

It would have been easy to add extra links and nodes to the hypertext graph in order to support this kind of navigation. For every message in a method node one can add a link to other method nodes. Some coding effort is needed to collect the pop-up menu from the anchors in the method and to open the new browser on the collection of valid methods, but in total this is feasible.
Apart from the extra resources involved (extra links when computing the contents a virtual method node) adding many links appeared to us an unnatural way of tackling the problem. An extension of the hypertext model was more appropriate, so we decided to build a QueryPath.
In contrast with a ContextPath, a QueryPath will not use contexts to resolve anchors. Instead it manages a set of <anchor, query> pairs: on activating an anchor the corresponding query is evaluated returning a context containing nodes and links. The path will pass this context to a special browser and open it.
As was the case with computational links, we used Smalltalk [SMALL] as the computation engine behind queries. Again the model is open enough to experiment with other languages here.

This prototype demonstrated the extensible nature of the architecture and proved that the model is able to handle hypertext structures without explicit links. The latter is due to the path concept.

4 PROTOTYPE II: AN ELECTRONIC AGENDA

To show the applicability of the model we implemented another application from an entirely different field, namely Computer Supported Co-operative Work (CSCW).
With others [HAL1, HAL2, CSCW] we believe that hypertext might play a substantial role in developing CSCW-applications. This persuaded us to favour an application from this domain as our second prototype. We picked an electronic agenda because we felt this could demonstrate the power of the hypertext model. We are well aware of the fact that this is not a very good candidate for investigating the possibilities of CSCW an sich [GRU1, GRU2, MAR].

What we are describing here is work in progress. For the moment the application runs on a single machine, and all agendas are kept in main memory. We will integrate database (Gemstone) and communication (plain e-mail) technology to build a real CSCW prototype. The user interface will be improved as well.

Figure 3 shows two open agenda managers. The window on the left is the one from 'Koen De Hondt': he has included Patrick Steyaert in the user list, so the bottom window contains agenda pages of both Koen and Patrick. On the right is the agenda of Karel Driesen with user list Karel, Koen and Serge. When Koen chooses 'Edit agenda ...' from the 'Agendas' menu, a dialogue window will appear where he can modify his agenda page. Afterwards the pages of both agenda managers will be updated to reflect the new status.

This application demonstrates the power of warm links and the added value when combining them with virtual nodes. The opening and closing of agendas is registered in an agenda server (for the moment a global variable, but in a later implementation it will be moved to the object oriented database). The agenda server is a hypertext node and the registering takes place via link pushing.
An agenda is modelled with a set of nodes containing the owner, the user-list, the date and the page. The page is a virtual node: it pulls information from the other nodes of the agenda and requests the page from the agenda-server (again by pulling links; attributes in the path are used as parameters).
When editing an agenda, the new data is pushed into the server. The server will push invalidation messages to all agendas opened on the same date and including this user in its user list. Recomputing the page node updates the window.

Besides the applicability of our approach, this experiment shows that the 'warm link' concept is useful, certainly when joined with virtual structures.

5 CONCLUSIONS & FUTURE WORK

We applied recent ideas from the world of software engineering to build a hypertext model supporting the idea of virtuality.
The major advantage of a virtual hypertext model is its open ended character. We have proven this by building a hypertext browser on top of a data structure without explicit hypertext concepts like links and nodes (i.e. a Smalltalk class browser and an electronic agenda).
Adding new concepts was relatively easy. This was experienced as a bonus, and was due to the extensible nature of an object oriented framework (the building methodology borrowed from the software engineering community).

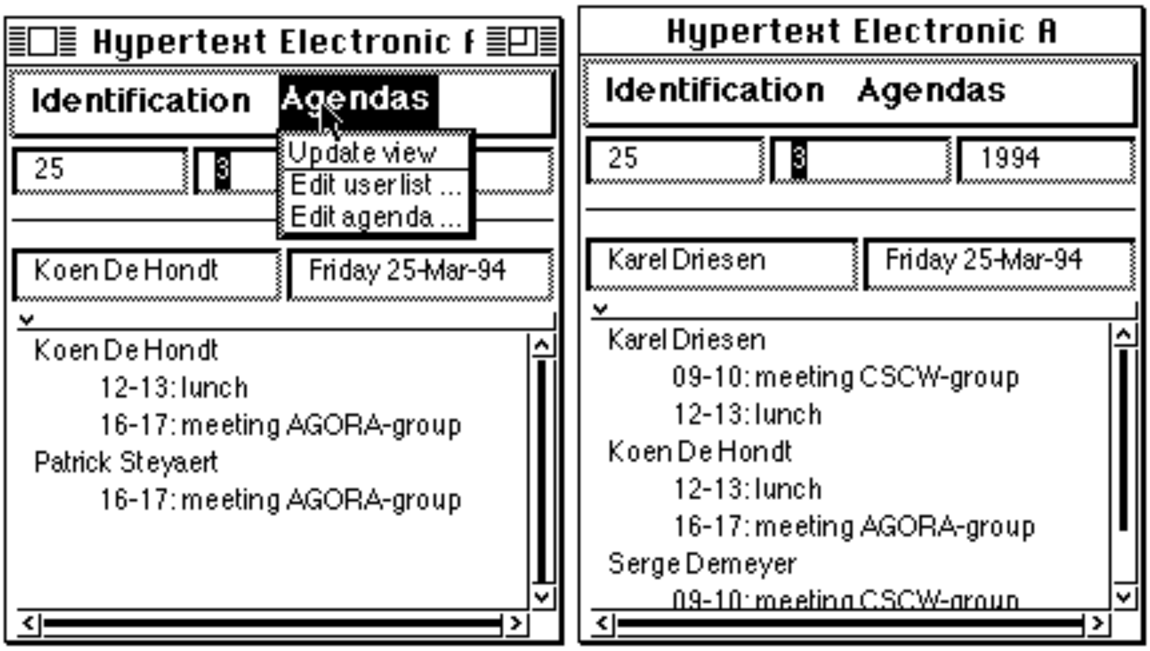

Figure 3: a view on two synchronising agenda managers

The model itself is a layered one. The first layer consists of the concepts Anchor, Node and Path; the notion of warm links [INTERMEDIA] is integrated. The second layer defines Contexts and Links as one of the possibilities to connect nodes.
It is rather new to have paths as first class citizens of a hypertext model, although Zellweger [ZEL] advocated this idea. It enabled us to experiment with the way nodes are connected and as such, makes links less important. To build computational power into the model (e.g. to make it virtual) we exploited the first class citizenship of paths and the warm linking facilities. The latter was needed to pass information over the network.
We perceived a conflict between the notion of computational links versus the notion of bi-directional links. We suggest two solutions: stick with unidirectional links and build an aggregation facility for links or limit the computational power of links to reversible calculations. We are able to conduct experiments in both directions, due to the extensibility of the framework and the ability to plug different computation engines into the model.

Iteration is known to be crucial in framework design [JO/RU]. As such we will continue to build applications on top of our class library, whenever necessary extending it to support emerging needs.
One particular class of applications we are thinking of is 'Computer Supported Co-operative Work' (CSCW). It is generally believed that hypertext technology can play a substantial role in developing CSCW applications [HAL1, HAL2, CSCW]. We plan to implement a 'Group Decision Support System' (GDSS) [GDSS] and see how this fits into our hypertext model.

ACKNOWLEDGEMENTS

I would like to thank Professor Theo D'Hondt for setting up the inspiring environment in which this work has found its roots. The various influences from the outside world have proven to be a fertile ground for a whole bunch of productive ideas.
All of my colleagues and all of the proof-readers are to be acknowledged here, but Patrick Steyaert (who provided the framework background), Koen De Hondt and Karel Driesen (for shooting holes while designing) deserve special attention.
Buon viaggio, mio caro, Ann.

REFERENCES

General References

[HT'87] ACM Hypertext'87 proceedings (November 13-15, Chapel Hill, North Carolina)
[HT'87+] ACM Hypertext'87 proceedings (November 13-15, Chapel Hill, North Carolina) + Communications of the ACM 31(7), July '88
[HT'89] ACM Hypertext'89 proceedings (November 5-8, Pittsburgh, Pennsylvania)
[NI'90] Proceedings of the 1990 NIST Hypertext Standardization Workshop (January 16-18, Gaithersburg, MD)
[EC'90] Rizk, A. / Streitz, N. / André, J. "Hypertext: concepts, systems and Applications - Proceedings of the European Conference on Hypertext" (November, Versailles, France)
[HT'91] ACM Hypertext '91 Conference Proceedings (December 15-18, San Antonio, Texas)
[EC'92] ACM Hypertext '92 Proceedings (November 30 - December 4, Milano, Italy)
[HT'93] ACM Hypertext '93 Conference Proceedings (November 14-18, Seattle, Washington USA)
[CA'94] Communications of the ACM 37(2), February '94

Specific References

[ABC] Schackelford, D. E., Smith, J.B. / Smith, F.D, "The Architecture and Implementation of a Distributed Hypermedia Storage System" in HT'93 [HT'93]
[ABC] Smith, J.B. / Smith, F.D, "ABC: A Hypermedia System for Artifact-Based Collaboration" in HT'91 [HT'91]
[AFR] Afrati, F. / Koutras, C.D. "A hypertext model supporting query mechanisms" in EC'90 [EC'90]
[AQU] Marshall, C. C. / Halasz, F. G. / Rogers, R. A. / Janssen, W. C. Jr. "Aquanet: A Hypertext Tool to Hold Your Knowledge in Place" in HT'91 [HT'91]
[AQU] Marshall, C. C. / Rogers, R. A. "Two Years Before the Mist: Experiences with AquaNet" in EC'92 [EC'92]
[AQU] Marshall, C. C. / Shipman, F. M. "Searching for the Missing Link: Discovering Implicit Structure in Spatial Hypertext" in HT'93 [HT'93]
[BEE] Beeri, C. / Kornatzky, Y. "A logical query language for hypertext systems" in EC'90 [EC'90]
[BIEB] Bieber, M. "Issues in Modeling a "Dynamic" Hypertext Interface" in HT'91 [HT'91]
[BRA] De Bra, p. / Houben, G. J. / Kornatzky, Y. "An Extensible Data Model for Hyperdocuments" in EC'92 [EC'92]
[CAS] Casanova, M. A. / Tucherman, L. / Lima, M. J. D. / Netto, J. L. R. / Rodriguez, N. / Soares, L. F. G. "The Nested Context Model for Hyperdocuments" in HT'91 [HT'91]
[CON] Conklin, J. "Hypertext: An Introduction and Survey" in IEEE Computer 20 (9), September 1987.
[COOM] Coombs, J. H. "Hypertext, Full Text, and Automatic Linking", Proceedings of SIGIR'90 - 13th conference on Research and Development in Information Retrieval (Brussels, Belgium)
[CRO] Crof, W. B. / Turtle, H. "A retrieval Model Incorporating Hypertext Links" in HT'89 [HT'89]
[CSCW] Streitz, N. / Halasz, F. / Ishii, H. / Malone, T. Neuwirth, C. / Olson, G. "The Role

of Hypertext for CSCW Applications (panel)" in HT'91 [HT'91]
[DEX] Halasz, F. / Schwartz, M. "The Dexter Hypertext Reference Model" in NIST'90 [NI'90]. An edited version of this paper appeared in [CA'94].
[DHM] Grønback, K. / Trigg, R. H. "Design issues for a Dexter-based hypermedia system" in EC'92 [EC'92]. Also in [CA'94]
[DHM] Grønback, K., Hem, J. A. / Madsen, O. L. / Sloth, L. "Designing Dexter-based Cooperative Hypermedia Systems" in HT'93 [HT'93]. Also in [CA'94]
[FREI] Frei, H.P. / Schäube, P. "Designing a Hypermedia Information System" in DEXA'91 Conference Proceedings (Vienna, Austria) (Springer-Verlag)
[FREI] Frei, H.P. / Stieger, D. "Making Use of Hypertext Links when Retrieving Information" in EC'92 [EC'92]
[GAM] Gamma, E. / Helm, R. / Johnson, R. / Vlissides, J. "Design patterns: Abstraction and Reuse of object-oriented design" in European Conference on Object-Oriented Programming Proceedings (July '93, Kaiserslauteren, Germany)
[GAR] Garg, P. J. "Abstraction Mechanisms in Hypertext" in HT'87 [HT'87+]
[GDSS] Kenis, D. / Bollaert, L. "MacPolicy: A Group Decision Support System" in Journal of Decision Systems-Revue des systeèmes de décision, numero special: Decision Support Systems: The IFORS-SPC 1 Conference 1992
[GDSS] Kenis, D. / D'Hondt, T. "A Client-server Architecture for Groupwork, Initial findings on the study of state-of-the-art computing environments for cross-disciplinary group dynamics" in EUC'92 Conference Proceedings (Apple's European University Consortium, April 21-23, Bruges, Belgium)
[GOO] Gyssens, M. / Paredaens, J. / Van Gucht D. "A Graph-Oriented Object Model for Database End-User Interfaces", ACM SIGMOD '90 Conference Proceedings (May 23-25, Atlantic City, New Jersey)
[GRU1] Grudin, J. "Groupware and Social Dynamics: Eight Challenges for Developers" in Communications of the ACM 37(1), January '94
[GRU2] Grudin, J. "Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces" in ACM CSCW'88 Proceedings (September 26-28, Portland, Oregon)
[GUI] Brown, P. J. "Turning Ideas into Products: The Guide System" in HT'87 [HT'87]
[HAL1] Halasz, F. "Reflections on NoteCards: Seven Issues for the Next generation of Hypermedia Systems" in HT'87 [HT'87+]
[HAL2] Halasz, F. "Seven issues revisited". Slides from the ACM Hypertext '91 Conference Keynote speech (December 15-18, San Antonio, Texas)
[HAM] Campbell, B. / Goodman, J. M. "HAM: A General - Purpose Hypertext Abstract Machine" in HT'87 [HT'87+]
[HDM] Caloini, A. "Matching Hypertext Models to Hypertext Systems: a Compilative Approach" in EC'92 [EC'92]
[HDM] Garzotto, F. / Paolini, P. / Schwabe "HDM: A Model for the Design of Hypertext Applications", D. in HT'91 [HT'91]
[HDM] Schwabe, D. / Cloini, A. / Garzotto, F. / Paolini, P. "Hypertext Development Using a Model-based Approach", Software-Practice and Experience, 22(11), November '92
[HELM] Helm, R. / Holland, I. M. / Gangopadhyay, D. "Contracts: Specifying Behavioral Compositions in Object-Oriented Systems" in ACM ECOOP/OOPSLA'90 Conference Proceedings (October 21-25, Ottawa, Canada)
[HFR] Wiil, U. K. / Legett, J. J. "Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems" in EC'92 [EC'92].
[HFR] Wiil, U. K. "Concurrency Control in Collaborative Hypertext Systems", ACM Hypertext '93 Conference Proceedings (November 14-18, Seattle, Washington USA).
[HPCD] Apple Computer, Inc. "Macintosh HyperCard User's Guide"
[HYD] Marmann, M. / Schlageter, G. "Towards a Better Support for Hypermedia Structuring: The HyDESIGN Model" in EC'92 [EC'92]
[INTERMEDIA] Catlin, T. / Bush, P. Yankelovich, N. "InterNote: Extending a Hypermedia Framework to Support Annotative Collaboration" in HT'89 [HT'89]
[INTERMEDIA] Haan, B. J. / Kahn, P. / Riley, V. A. / Coombe, J. H. / Meyrowitz, N. K. "IRIS Hypermedia services" in Communications of the ACM 35(1), January '92
[INTERMEDIA] IRIS (Institute for Research in Information and Scholarship) "IRIS Intermedia User's Guide". User's guide with InterMedia 3.0 (Brown University)
[INTERMEDIA] Yankelovich, N. / Meyrowitz, N. / Van Dam, A. "Reading and Writing the Electronic Book" in IEEE Computer, October '85
[JO/FO] Johnson, R. E. / Foote, B. "Designing Reusable Classes" in Journal of Object-Oriented Programming 1(2), February '88 p. 22 - 35
[JO/RU] Johnson, R. E. / Russo, V. F. "Reusing Object-Oriented Designs" University of Illinois technical report UIUCDCS 91-1696
[KMS] Ackscyn, R. / McCracken, D. / Yoder, E. "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations" in HT'87 [HT'87+]
[LAN] Lange, D. B. "A Formal Model of Hypertext" in NIST'90 [NI'90]
[LUC] Lucarella, D. "A model for hypertext-based information retrieval" in EC'90 [EC'90]
[MAR] Markus, M. L. / Conolly, T. "Why CSCW applications fail: Problems in the adoption of interdependent work tools" in ACM CSCW'90

Proceedings (October 7-10, Los Angeles, California)
[MED1] Frisse, M. E. "Searching for Information in a Hypertext Medical Handbook" in HT'87 [HT'87]
[MED2] Frisse, M. E. / Cousins, S. B. "Information Retrieval from Hypertext: Update on the Dynamic Medical Handbook Project" in HT'89 [HT'89]
[MEYR] Meyrowitz, N. "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework" in ACM OOPSLA'86 Conference Proceedings (Sept 29 - October 2, Portland, Oregon)
[MICR] Fountain, A. / Hall, W. / Heath, I. / Davis, H.C. "MICROCOSM: An open model for hypermedia with dynamic linking" in EC'90 [EC'90]
[MIX1] Bracha, G. / Cook, W. "Mixin-based inheritance" in ACM ECOOP/OOPSLA'90 Conference Proceedings (October 21-25, Ottawa, Canada)
[MIX2] Steyaert, P. / Codenie, W. / D'Hondt, T. / De Hondt, K. / Lucas, C. / Van Limberghen, M. "Nested Mixin-Methods in Agora". Proceedings of ECOOP'93 (7th European Conference on Object-Oriented Programming; July 26-30, Kaiserslautern, Germany). Springer-Verlag Lecture Notes in Computer Science no. 707
[MIX2] Boyen, N. / Lucas, C. / Steyaert, P. "Generalised Mixin-based Inheritance to Support Multiple Inheritance". Submitted to ACM OOPSLA '94 Conference. Will be available through anonymous ftp at 'progftp.vub.ac.be'
[MUL] Rizk, A. / Sauter, L. "Multicard: An Open Hypermedia System" in EC'92 [EC'92]
[MWB] Nanard, J. / Nanard, M. "Should Anchors Be Typed Too ? An experiment with MacWeb" in HT'93 [HT'93]
[MWB] Nanard, J. / Nanard, M. "Using Structured Types to Incorporate Knowledge in Hypertext" in HT'91 [HT'91]
[NEP] Delisle, N. / Schwartz, M. "Neptune: a Hypertext System for CAD applications", ACM SIGMOD '86 Conference Proceedings (28-30 May, Washington, DC)
[NIELS] Nielsen, J. "Hypertext & Hypermedia" (Academic Press)
[PAR] Van Dyke Parunak H. "Don't Link Me In: Set Based Hypermedia for Taxonomic Reasoning" in HT'91 [HT'91]
[PAR] Van Dyke Parunak H., "Hypercubes Grow on Trees (and Other Observations from the Land of Hypersets)" in HT'93 [HT'93]
[PATT] Gamma, E. / Helm, R. / Johnson, R. / Vlissides, J. "A Catalog of Object-Oriented Design Patterns" Draft verion of a report to be submitted for publication.
[PHI] McCall, R. / Bennet, P. / d'Ornozio, P. D. / Ostwald, J. / Shipman, F. / Wallace, N. "PHIDIAS: integrating CAD-graphics into dynamic hypertext" in EC'90 [EC'90]
[PINT] Pintado, X. / Tsichritziz, D. "Satellite: Hypertext navigation by affinity" in EC'90 [EC'90]
[QRL] Charoenkitkarn, N. / Tam, J. / Chignell, M. H. / Golovchinsky, G. "Browsing Through Querying: Designing for Electronic Books" in HT'93 [HT'93]
[SCHU] Schütt, H. / Streitz, N. "HyperBase: A hypermedia engine based on a relational data-base management system" in EC'90 [EC'90]
[SCH] Schnase, J. L. / Leggett, J. J. "Computational Hypertext in Biological modelling" in HT'89 [HT'89]
[SMALL] Golberg, A. / Robson, D. "Smalltalk-80: The Language and its Implementation" (Addison Wesley)
[SMI] Smith, K. E. / Zdonik, S. B. "Intermedia: A case study of the Differences Between Relational and Object-Oriented Database Systems" in ACM OOPSLA'87 Conference Proceedings (October 4-8, Orlando, Florida)
[SUP] Egan, D. E. / Lesk, M. E. / Ketchum, R. D. / Lochbaum, C. C. / Remde, J. R. / Littman, M. / Landauer, T. K. "Information Retrieval from Hypertext: Update on the Dynamic Medical Handbook Project" in HT'91 [HT'91]
[SUP] Remde, J. R. / Gomez, L. M. / Landauer, T. K. "Superbook: An automatic Tool for Information Exploration - Hypertext ?" in HT'87 [HT'87]
[TRE] Furuta, R. / Stotts, P. D. "Programmable Browsing Semantics in Trellis" in HT'89 [HT'89]
[TRE] Furuta, R. / Stotts, P. D. "The Trellis Hypertext Reference Model" in NIST'90 [NI'90]
[TRE] Stotts, P. D. / Furuta, R. / Ruiz, J. C. "Hyperdocuments as Automata: Trace-based Browsing Property Verification" in EC'92 [EC'92]
[TRE] Stotts, P. D. / Furuta, R. "Hierarchy, composition, scripting languages, and translators for structured hypertext" in EC'90 [EC'90]
[TRE] Stotts, P. D. "Dynamic Adaption of Hypertext Structure" in HT'91 [HT'91]
[UNI] Abelson, H. / Sussman, G. J. "Structure and Interpretation of Computer Programs" sections 4.2-4.5 (The MIT Press)
[VOL] Bernstein, M. / Bolter, J. D. / Joyce, M. / Mylonas, E. "Architectures for Volatile Hypertext" in HT'91 [HT'91]
[WALT] Frisse, M. E. / Cousins, S. B. / Hassan, S. "WALT: A Research Environment for Medical Hypertext" in HT'91 [HT'91] (Technical briefing)
[WEB] Monnard, J. / Pasquier-Boltuck, J. "An Object-Oriented Scripting Environment for the WEBSs Electronic Book System" in EC'92 [EC'92]
[WWW] NCSA-MOSAIC. To get information send e-mail to mosaic@ncsa.uiuc.edu. Mosaic is available through anonymous ftp at 'ftp.ncsa.uiuc.edu'
[YOU] De Young, L. "Linking considered harmful" in EC'90 [EC'90]
[ZEL] Zellweger, P. T. "Scripted Documents: A Hypermedia Path Mechanism" in HT'89 [HT'89]

[ZEL] Zellweher, P. T. "Position Statement in Structure, Navigation, and Hypertext: The Status of the Navigation Problem" in HT'91 [HT'91]