# Schema-based authoring and querying of large hypertexts*

Bernd Amann, Michel Scholl

*Cedric CNAM, 292 Rue St. Martin, 75141 Paris Cedex 03 France*
*INRIA-Rocquencourt, 78153 Le Chesnay Cedex France*
*email : amann@cnam.fr, scholl@cnam.fr*

Antoine Rizk

*Euroclid, 12 Avenue des Près, 78180 Montigny le Bretonneux France*
*email : Antoine.Rizk@inria.fr*

**Abstract**

Modern hypertext applications require new system support for hypertext authoring and user navigation through *large* sets of documents connected by links. This system support must be based on advanced, typed data models for describing the information structure in different application domains. Schema based structuring through strongly typed documents and links has already been proposed and put to practical use in a multitude of hypertext applications. Systems such as Multicard/$O_2$ and MORE have moreover exploited conceptual schemas for querying the resulting hyperdocuments in a more structured way. In this paper, we show how hypertext schemas and query languages can be utilized for designing hypertext authoring and browsing environments for large hypertexts. We illustrate our mechanisms using the Gram data model and describe their implementation on top of the Multicard hypermedia system connected to the $O_2$ object-oriented database management system.

## 1 Introduction

Managing large-scale hypertext applications as well as simplifying navigation in large hypertexts are key issues for the next generation of hypertext systems. Traditionally, hypertext systems are viewed as networks of documents where the typical way of accessing data is based on step-by-step navigation by following links from one document to the other. Users not only traverse links without prior indication to their meaning, but also can add new documents and links without necessarily specifying their semantics. This flexibility, which is due to the absence of structure in hypertext information, is at the origin of the success of such systems.

Basic hypertext browsing, i.e. following links one by one, however, is not sufficient for exploring large hypertext networks efficiently. User disorientation and cognitive overhead (Utting and Yankelovich 1989) are two well known problems due to the size of most hypertexts. In answer to these problems, current hypertext systems implement a variety of additional navigation mechanisms such as graphical browsers, fish-eye views, guided tours and backtracking. These tools are put at the disposal of the readers but they are, in general, inaccessible to the authors for tailoring to specific application domains.

A first solution to this limitation were procedural script languages such as Hypertalk (Apple Computer Inc. 1987), MultiTalk (Rizk and Sauter 1992) or WebTalk (Nanard and Nanard 1991). These languages are based on control structures and commands for navigating in the network, investigating node and link attributes, defining guided tours or synchronizing external events. Whereas this technique allows the *procedural* description of rather complex navigation semantics, its application is limited to small hypertext networks. In order to manipulate large sets of nodes and links in a coherent way, authors need more *declarative* models and description languages.

We argue that hypertexts should be considered as typed semi-structured documents[1]. Typing usually implies the classification of documents into relations or classes specified by a few attributes. Such attributes can express a document semantics which is more or less related to its contents. By storing structured information

*This article is an extended version of *Querying Typed Hypertexts in Multicard/$O_2$, ACM European Conference on Hypermedia Technology (ECHT'94), Edinburgh, Scotland, September 1994*

[1]Being sets of documents, they should be managed and queried on their *contents* using information retrieval and full-text indexing techniques. We shall not consider this issue in this paper.

about documents, and possibly documents as well, in databases, it becomes possible to take advantage of database technology in terms of persistence, versioning, distribution, transaction management, concurrency control and recovery, fast access by structure, etc. In addition to these benefits, document structuring is a tool for better conception (strong typing of documents is a means of constraining the author's choices), organization, maintenance and evolution of documents. However, this does not imply that hypertext systems should be viewed as another application of database systems since they keep their specificities, among which access by contents or by step-by-step navigation through links are the most important ones.

This need for (1) a better hypertext structuring and for (2) an integration with database systems has already been widely recognized in the past few years (see for example (Rossiter, Sillitoe, and Heather 1990)). The emphasis of this paper is on the first requirement. Our objective is to demonstrate and to validate through an implementation the use of declarative database modeling and querying for developing general query based user navigation tools. In particular typed paths in the network of documents are expressed by database queries. This has several simple applications such as restricting navigation to those paths in the hypertext which satisfy a query.

By integrating hypertexts and database concepts (schema, query language), a new class of users emerges which is that of *designer* (or analyst). We provide a set of basic mechanisms that could serve as a generic platform for authors and designers to develop navigation tools for their specific application domains. The role of the designer consists of (1) identifying the application basic type constituents, (2) creating the hypertext schema to which the author has to adhere, and (3) providing the interface and tools, based on the mechanisms we will describe, for the author to use when creating his navigation patterns. The role of the author is the same as for classical hypertexts except that, in our case, he/she is restrained by the system to abide by the rules defined by the designer. The author has at his disposal a rich set of menus, buttons and tools, created by the designer for the particular application domain. Finally, the reader can use a set of graphical and interactive interface tools for navigating and creating queries based on the schema.

A fundamental underlying assumption to our work is of course the fact that typing is both a good structuring mechanism that results in better authoring, as well as a support for structured querying that results in better authoring and reading. A direct effect of this assumption is that for the result to be useful to readers, a significant effort must be put into authoring/design. In the CSCW (Computer Supported Collaborative Work) field, this mismatch between who does the extra work (the author) and who gets the benefit (the reader) is considered a potential reason for failure (Grudin 1994). Contrary to CSCW, for hypertext systems we argue that this mismatch in effort due to typing is the only way to resolve the management of large hypertext applications, for many reasons:

- Typing has been proven in hypertext as an aid for structuring (Nanard and Nanard 1991; Garzotto, Paolini, and Schwabe 1993).
- Querying in hypertext has often been stated as a requirement (Halasz 1988; Consens and Mendelzon 1989; Beeri and Kornatzky 1990; Amann and Scholl 1992; Lucarella, Parisotto, and Zanzi 1993). Typing in our model is an aid not only to structuring, but also to querying. We admit that this could be even more effective if coupled to information retrieval by contents, but it is not in the scope of our work.
- In addition to the classical hypertext author/reader model, we introduce the designer/analyst role who is in fact the one who really takes on the extra effort. The author's role in our model is simplified because the designer will put at his disposal a set of domain tailored authoring tools and mechanisms.
- The role of the designer taking on the extra effort from readers is a validated common practice in database systems.

This kind of problems however must not be ignored at the application design, which should result in a tailorable environment providing flexibility to both authors and readers.

In the rest of this paper, we will present a platform for implementing advanced authoring and browsing mechanisms in large hypertexts. We use the Gram data model (Amann and Scholl 1992) for the presentation of these mechanisms, because we believe it is well adapted to the expression of typed paths in the hypertext. These mechanisms can nevertheless be expressed in other query languages with similar expressive power. In particular, relational or object-oriented query languages are good candidates.

In Section 2, we give a short review of the Gram data model through examples of data and queries. Section 3 is the main contribution of this paper and describes the application of query languages to hypertext authoring. We introduce a few basic mechanisms, namely virtual link, reverse link and structural backtracking as new interpretations of structural queries in hypertexts. A visual query interface that allows naive readers to formulate ad hoc queries by following paths in graphical schema representations is presented in Section 4. Section 5 introduces the concept of "zooming" as a new information structuring mechanism for making hypertext systems cooperate in a transparent way. Finally, Section 6 describes the feasibility of our approach through an implementation with the $O_2$ Database Management System (DBMS).

## Related Work

Typing has appeared for the first time in hypertext systems for specific application domains such as collaborative work. In these medium-sized *collaborative* hypertext applications (Rada 1991), discussion, authoring and annotating are supported by powerful user interfaces and authoring environments. Privacy, messaging, and synchronization are other important issues addressed by these systems. The gIBIS (graphical IBIS) (Conklin and Begeman 1988) hypertext system supports collaborative system design and discussion by providing a fixed set of node and link types defined by the IBIS (Issue Based Information Systems) design method. The storage manager is implemented on top of a relational DBMS and allows to select nodes according to simple predicates on their contents and attribute values. The cooperative hypermedia authoring environment SEPIA (Structured Elicitation and Processing of Ideas for Authoring) (Streitz, Hannemann, and Thüring 1989) supports both individual and cooperative writing of hypermedia documents. The design space is divided into several activity spaces providing specific design objects and operations according to the authors activities. In the planning space, authors can for example control their activities by developing issue structures based on IBIS. The Aquanet (Marshall, Halasz, Rogers, and Janssen 1991) system is a more general platform which allows designers to customize knowledge structures by the definition of specific node and link types. A graphical browser-based user interface can be used to create, compose and modify large graphical knowledge structures.

Typing has also been used for a better representation of *rich* hypertext structures. These systems take advantage of typed nodes and links for the creation of flexible user interfaces and allow authors to define different semantic views (conceptual documents) of the same underlying hypertext network. MacWeb (Nanard and Nanard 1991) uses structured types and the WebTalk script language for defining generic document models (conceptual documents) which are instantiated according to the underlying hypertext network. For each document, the author can define the logical structure, the presentation and the modification behavior by using selection, presentation and interaction rules. A similar approach is proposed in the HyperPro (Osterbye and Normark 1994) prototype which introduces the notion of interaction schema for defining and controlling the interaction on typed hypertext networks.

An important step towards general typed hypertext models has been the design of *large* hypertext applications according to hypertext schemas that describe some semantics of the application domains. In HDM (Hypertext Design Model) (Garzotto et al. 1993), application schemas describe overall classes of information sources (node types) in terms of their common presentation characteristics, their internal organization structure and their mutual interconnections (link types).

All of the above systems introduce strong typing as an authoring and modeling tool without using the underlying structure for formulating queries. Nevertheless, hypertext querying has been an evolving research topic since the early systems. In (Marchionini and Shneiderman 1988), browsing and searching were observed as complementary user interaction styles for powerful user interface models.

Standard *information retrieval* techniques have been adapted to the hypertext context in systems such as the Medical Handbook (Frisse 1988), $I^3R$ (Croft and Turtle 1989) and Intermedia (Coombs 1990). The document base consists of a set of documents each of which has an associated set of links and an information retrieval style description (Dunlop and Rijsbergen 1991). Hypertext links are mainly used to improve the description of documents and the presentation of search results.

A different approach is necessary for querying hypertext networks with respect to their *structure*. The hypermedia system PHIDIAS (McCall, Bennett, D'Oronzo, Ostwald, Shipman, and Wallace 1990) supports computer-aided design based on CAD graphics and IBIS argumentation. The core of PHIDIAS is an augmented version of MIKROPLIS (McCall, Mistrik, and Schuler 1980), one of the first hypertext systems allowing both content- and structure based search. The query language can be used interactively and for defining virtual (dynamic) structures : queries are embedded in nodes and evaluated when the nodes are displayed. In the object-oriented hypermedia system Panorama (Chen, Ekberg, and Thompson 1990), an information navigator allows users to create typed hypertexts and specify queries by using a menu-based natural language interface. These queries are translated into the query language provided by the underlying object-oriented database management system Zeitgeist (Ford, Joseph, Langworthy, Lively, Pathak, Perez, Peterson, Sparacin, Thatte, Wells, and Agarwal 1988).

The following *graph-based models and query languages* are to our knowledge the first systematic uses of database modeling, not only as an authoring tool, but also as a reader query based navigation tool. All of these approaches (Consens and Mendelzon 1989; Afrati and Koutras 1990; Beeri and Kornatzky 1990; Amann and Scholl 1992; Lucarella et al. 1993) use queries for selecting (typed) paths in the graph of documents. Compared to other approaches using SQL and relational algebra (Gallagher, Furuta, and Stotts 1990), they are more powerful since they allow some recursion in paths. However, the importance of path recursion in hypertext navigation has never been precisely stated and quantified.

Among the above graph-based models, GraphLog (Consens and Mendelzon 1989) is noteworthy. It is a visual query language for graph based information. Queries are formulated by drawing graph patterns which are then matched against the underlying hypertext graph. Paths are described by regular expressions on node and link labels. Whereas it has been shown that GraphLog is well adapted to the formulation of structured hypertext queries, it is essentially a graphical user interface, where some queries can easily be expressed by naive users, while the formulation of others remain complex.

The MORE (Multimedia Object Retrieval Environment) (Lucarella et al. 1993) system supports the *direct* manipulation of multimedia information which is described by a graph-based object model. A visual user interface allows designers and readers to define conceptual schemas, visualize object instances and formulate queries by manipulating object graphs on the screen. Whereas direct object manipulation is realized by two basic operations for viewing and browsing the results of ad-hoc queries, the query language is not part of the authoring environment itself.

Gram (Amann and Scholl 1992) is described in the following section. Similar to GraphLog and various object-oriented query languages (e.g. the standard OQL (ODMG 1994)), Gram is based on path expressions. Although different from GraphLog, the path expressions of Gram are also described by regular expressions on the node and link types. Its SQL like syntax is convenient for the reader familiar with SQL. It uses regular expressions to describe the query range (FROM clause) which is well adapted to the user in the presence of a schema graph (see below). A graphical query interface under development is a first step toward a complete visual query language and tool for query based navigation through hypertexts.

# 2 Preliminaries

We first give a brief review of the Gram model which serves as the underlying data model for the description of the mechanisms presented in Section 3. For a formal presentation of Gram, see (Amann and Scholl 1992).

## 2.1 Hypertext Schema

A hypertext schema is a directed labeled graph. Nodes represent document types and edges correspond to the possible types of links between documents. Figure 1 shows the schema of a MOVIE GUIDE hypertext storing multimedia information about movies and artists.

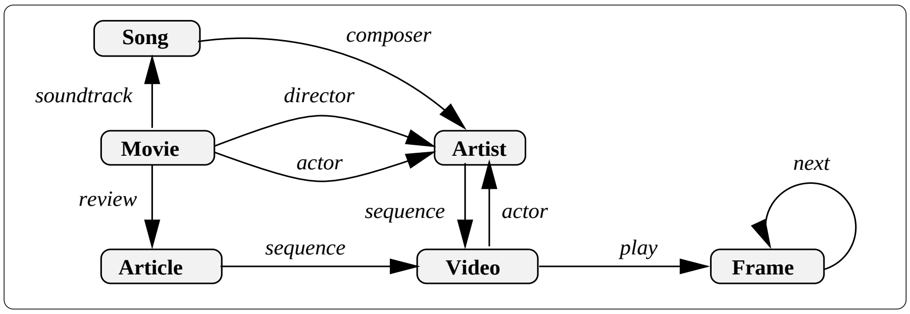


Figure 1: Gram Schema for the MOVIE GUIDE hypertext

Each document has one of the types `Song`, `Article`, `Movie`, `Artist`, `Video` and `Frame`. Movies are connected to their directors and actors by links of type `director` or `actor` respectively. Review articles about movies are accessible via links of type `review`. These articles and the documents about artists (directors, actors and composers) might refer to specific video sequences through links of type `sequence`. In fact, running a video corresponds to the selection of a link of type `play` followed by the automatic traversal of `next` links connecting the different video frames. This kind of automatic guided tour can be used to model, and navigate within, active media such as video and sound. Actors appearing in videos are referenced by links of type `actor`. Movie soundtracks can be listened to by following links of type `soundtrack`. They are connected to their composers by links of type `composer`. Note that links may contain information such as an actors role (type `actor`) or the starting and ending frame of a video sequence (type `play`).

## 2.2 Hypertext Query Language

The Gram query language is based on a path algebra and makes it possible to (1) select paths in hypertext networks according to regular expressions over document and link types, (2) select paths according to document and link values, (3) construct new paths by projection and concatenation, (4) construct connected subgraphs by joining paths sharing documents. For illustration, assume a reader is interested in horror movie pictures and wants to get documents about their soundtracks and composers. The following answers his query[2] :

```
SELECT Song composer Artist
  FROM Movie soundtrack Song composer Artist
 WHERE Movie.genre = ''horror''
```

Paths satisfying the regular expression `Movie soundtrack Song composer Artist` (`FROM`-clause) and the `WHERE`-clause condition are selected and projected on the regular expression `Song composer Artist` (`SELECT`-clause).

The answer to the following query contains all review articles of movies directed by Ingmar Bergman. Note that any pair of paths $p_1 \models$ `Movie director Artist` and $p_2 \models$ `Movie script Article` in the range of the query have to share the document of type `Movie` :

```
SELECT Article
  FROM Movie director Artist + Movie review Article
 WHERE Artist.name = ''Ingmar Bergman''
```

The relational algebra or SQL would be powerful enough to express Gram queries without Kleene closure (recursion) by using natural join for navigating in the relational schema. On the contrary, the following query requires recursion capabilities and returns all video frame sequences of movie pictures directed by Alfred Hitchcock :

```
SELECT (Frame next)* Frame
  FROM Movie review Article sequence Video play (Frame next)* Frame
                           + Movie director Artist
 WHERE Artist.name = ''Alfred Hitchcock''
```

Kleene closure introduces a restricted form of recursion into the Gram algebra. Gram has less expressive power than first-order logic with transitive closure (FO+TC) and therefore than GraphLog (Consens and Mendelzon 1990) since we restrict hypertexts to be connected graphs (Amann 1994). However, Gram queries can keep the whole information about paths of variable length, while first-order formulas with transitive closure only return the extremities of recursive path.

# 3 Query-Based Hypertext Authoring

This section is devoted to the application of query languages to hypertext authoring and describes a few basic query-based mechanisms for implementing a variety of hypertext navigation patterns in a declarative way.

We assume a multi-window user interface where several documents can be displayed at the same time. Without querying capabilities, navigation consists of clicking on a button $\boxed{\rightarrow, l}$ in a displayed source document $s$, which results in the traversal of a binary link $l = (s,t)$ and the display of the target document $t$. Without loss of generality, we assume a global navigation context including a set of displayed documents $\mathcal{D}$ and a navigation history stack $\mathcal{H}$. Whenever a link $l = (s,t)$ is followed from some source document $s \in \mathcal{D}$, the target document $t$ is displayed on the screen and pushed on the navigation history stack $\mathcal{H}$. At the same time, in order to restrict the number of displayed documents, the window of the source document is deleted, i.e. removed from $\mathcal{D}$. More formally, we define the following navigation operation that is executed by clicking on a button $\boxed{\rightarrow, l}$, $l = (s,t)$ :

$$next(l) : \left[ \begin{array}{ll} \mathcal{D} = \mathcal{D} \cup \{t\} & \textit{display target document} \\ push(\mathcal{H}, t) & \textit{push target on history stack} \\ \mathcal{D} = \mathcal{D} - \{s\} & \textit{undisplay source document} \end{array} \right.$$

Step 3, i.e. the undisplay of the source document, might be inappropriate in situations such as, for example, when the user wants to see two documents side by side. A solution to this is to attach distinct behaviors to the different mouse buttons (left, middle, right) selected by the user. As it is the case for some hypertext interfaces, the left button might replace the displayed document, whereas the middle button might open a new window without replacement.

In order to backtrack navigation history and to return to some previous navigation context, readers might click on a "back" button $\boxed{\hookleftarrow}$ and repeat the following navigation operation :

[2] Whereas an SQL like syntax is used for clarity, one has to observe that the underlying data model defines a path algebra on typed graphs.

$$back : \left[ \begin{array}{ll} d = pop(\mathcal{H}) & \textit{pop last document} \\ d' = pop(\mathcal{H}) & \textit{pop previous document} \\ \mathcal{D} = \mathcal{D} \cup \{d'\} & \textit{display previous document} \\ push(\mathcal{H}, d') & \textit{push previous document on history stack} \\ \mathcal{D} = \mathcal{D} - \{d\} & \textit{undisplay last document} \end{array} \right.$$

In the following, we will show how conceptual schemas and query languages can support a number of homogeneous navigation mechanisms and browsing facilities in large hypertext networks.

### 3.1 Virtual Links

A number of existing hypertext systems support advanced information retrieval and indexing mechanisms for the definition of virtual or intensional links (DeRose 1989) between documents. In contrast to extensional "hard-wired" links, virtual links are the result of queries on the hyperdocument contents and structure. Independently of the underlying data model, queries are supposed to return sets of paths and documents $d$ and $d'$ are said to be connected by a virtual link if there is a path $p$ from $d$ to $d'$ in the hypertext network.

More formally, a virtual link, defined by a (path) query $Q$ and denoted $link(Q)$, is the set of node (document) couples connected by paths in Q :

$$link(Q) = \{(d_1, d_k) | d_1 \dots d_k \in Q\}.$$

While $Q$ brings complete information on its paths, $link(Q)$ only keeps the paths end-nodes. A virtual link $link(Q)$ is followed from some document $s$ by the following operation :

$$next(s, link(Q)) : \left[ \begin{array}{ll} T = \{t | (s,t) \in link(Q)\} & \textit{get target documents} \\ \mathcal{D} = \mathcal{D} \cup T & \textit{display target documents} \\ push(\mathcal{H}, T) & \textit{add target documents to history} \\ \mathcal{D} = \mathcal{D} - \{s\} & \textit{undisplay source document} \end{array} \right.$$

Instead of opening all target documents $T$ simultaneously, some control strategy has to be used to avoid an "explosion" of the number of windows displayed on the screen. For example, if the number of documents in $T$ is less than some limit, say 4, all documents are opened and displayed simultaneously. Otherwise, the user can choose the documents to be opened from a menu showing all document titles.

Virtual links introduce flexibility and allow the control of the hypertext size by avoiding unnecessary extensional links between documents. Note also that $next(d, link(Q))$ might be precomputed for faster navigation – which implies recalculation when the hypertext graph is modified – and that this mechanism could be implemented in a relational model by a view mechanism (as long as $Q$ does not contain any Kleene closure).

**Example 3.1** *In order to play the stored video sequences of the movie* Jurassic Park, *a reader can first display all review articles by successively selecting links of type* `review` *and then follow links of type* `sequence` *to the specific videos. A better solution is to use the following query Q defining a virtual link* link(Q) *between the* `Movie` *document with the title* `Jurassic Park` *and all video sequences that can be reached by paths satisfying the regular expression* `Movie review Article sequence Video` *:*

```
SELECT Video
  FROM Movie review Article sequence Video
 WHERE Movie.title = ''Jurassic Park''
```

*By clicking on a new button* $\boxed{\rightarrow, Q}$ *attached to the previous query $Q$, readers might then directly display all* `Video` *documents containing video sequences of the movie Jurassic Park. Instead of selecting the source document explicitly by the condition* `Movie.title = ''Jurassic Park''`*, an author could specify the following "generic" query, where the formal parameter* `Active Document` *is replaced by the active document, i.e. the document with the button* $\boxed{\rightarrow, Q}$ *:*

```
SELECT Video
  FROM Movie review Article sequence Video
 WHERE Movie = Active Document
```

*Afterwards, the same query can be reused for creating virtual links between any movie and its video sequences. This example also illustrates how virtual links prevent unnecessary navigation by following several paths to the same document. Although there might exist different review articles pointing to the same video, the latter appears only once in the result of the query.*

Hypertext browsing implies an asymmetric view of data since documents are accessed through *directed* paths in a graph. However, this asymmetry is only apparent and we can always obtain the set of starting nodes of paths satisfying some query $Q$ and leading to a given node. This corresponds to the projection of the paths in $Q$ on their source nodes. A virtual link $link(Q)$ is traversed from some document $t$ in the reverse direction by the following operation :

$$previous(t, link(Q)) : \left[ \begin{array}{ll} T = \{s|(s,t) \in link(Q)\} & \textit{get source documents} \\ \mathcal{D} = \mathcal{D} \cup T & \textit{display source documents} \\ push(\mathcal{H}, T) & \textit{add source documents to history} \\ \mathcal{D} = \mathcal{D} - \{t\} & \textit{undisplay target document} \end{array} \right.$$

As for "forward" links this mechanism can be implemented either by storing $Q$ with a button $\boxed{\leftarrow, Q}$ or by ad-hoc querying.

## 3.2 Dynamic Guided Tours

Navigation, even when query assisted, becomes extremely complex when the number of nodes and edges in the hypertext network passes some limit. Navigation space restriction is then necessary. Several mechanisms such as guided tours (Trigg 1988; Marshall and Irish 1989) or active paths (Zellweger 1989) have been proposed in literature. We suggest to implement such mechanisms by restricting navigation to paths returned by a query $Q$. Then readers navigate step-by-step but only in the predefined space. Only buttons corresponding to the paths of $Q$ are active in each document, the others being disabled. In some cases, for example when the author only wants to suggest a guided tour to the readers, the corresponding buttons might be signalized by a special color or font without deactivating the buttons outside the guided tour. In that case, readers can follow the proposed paths but also start side-trips for exploring new parts of the network and, if required, easily return to the guided tour. As for virtual links this mechanism can be implemented either by storing $Q$ with a button $\boxed{\otimes, Q}$ or by ad-hoc querying.

## 3.3 Structured Backtracking

Most hypertext systems implement a stack based history model and allow readers to return to previously visited documents by using a special back button $\boxed{\hookleftarrow}$. More sophisticated mechanisms distinguish between several types of traversal history logs and backtracking strategies (e.g. chronological and task-based) (Bieber and Wang 1994). Conditional or parametric backtracking (Garzotto, Mainetti, and Paolini 1993) evaluates query expressions in the navigation history log and returns to the most recently displayed document satisfying the queries.

Virtual links as defined above obviously are insufficient for implementing history mechanisms. For example, after having followed a link of type `director` from the movie *Le Dernier Metro* to the document about François Truffaut, the activation of the button $\boxed{\leftarrow, \texttt{Movie director Artist}}$ in this document does not return only to the movie *Le Dernier Metro*, but to all movies directed by François Truffaut.

A (path) query language can be used for implementing *structured backtracking*. Instead of following the history step-by-step (button $\boxed{\hookleftarrow}$), a whole path returned by a query might be "popped" from the history stack : a path $p$ can be *popped* from a history stack $\mathcal{H}$, denoted $pop(p, \mathcal{H})$, if all documents in $p$ have been visited in the same order as they occur in $p$. A number of different backtracking strategies are possible :

- A query might contain several candidate paths that can be popped from the history. Different solutions might be proposed where it is for example possible to select the shortest (longest) path automatically or to let the reader choose a path in a menu and, in this way, influence the backtracking strategy.
- A path might be popped from the history stack if the reader has followed all its links one by one, but also if (2) all nodes have just been visited in the same order. The second strategy is less restrictive and accepts "side-trips" to documents that do not occur in the selected path.

As we have done for the definition of virtual links, we can use a back button attached to a query $Q$ : by clicking on the button $\boxed{\hookleftarrow, Q}$, the starting document of the selected path $p$ in $Q$ will be displayed.

**Example 3.2** *A reader who watches a video sequence of a movie, i.e. follows a path satisfying the regular expression* `Movie review Article sequence Video play Frame (next Frame)*`, *can interrupt the video display and return to the review article by clicking on the button* $\boxed{\hookleftarrow, \texttt{Article sequence Video play Frame (next Frame)*}}$.

## 4 Schema Browsing

We have illustrated how hypertext authors can prepare virtual links and dynamic guided tours by attaching (path) queries to buttons. A different solution is to provide readers with a high-level user interface for ad-hoc querying during hypertext browsing. Such an interface can be implemented in the form of a schema browser where users (authors and readers) select node and link types in order to create path expressions in an interactive style. In this section, we describe the general functionalities of a schema browser that we have implemented on top of the Multicard/$O_2$ hypermedia database prototype (Section 6). Hypermedia applications are built by using a hypermedia toolkit, a script language and the schema browser.

The schema browser is composed of three windows containing (1) a visual representation of the hypertext schema graph, (2) the active document (node) and (3) the generated query that can be used ad-hoc or stored in a script, attached to a button in the active document.

**Example 4.1** *An example of the ad-hoc definition of a reverse link by using the schema browser is shown in Figure 2. In order to display all cinemas featuring a movie directed by Alfred Hitchcock, the user has activated the document* Alfred Hitchcock *and selected the link types* `show` *and* `director` *in the schema graph representation of the movie guide. This graphical selection is equivalent to the following Gram query :*

```
SELECT Cinema
  FROM Cinema show Movie director Artist
 WHERE Artist = Active Document
```

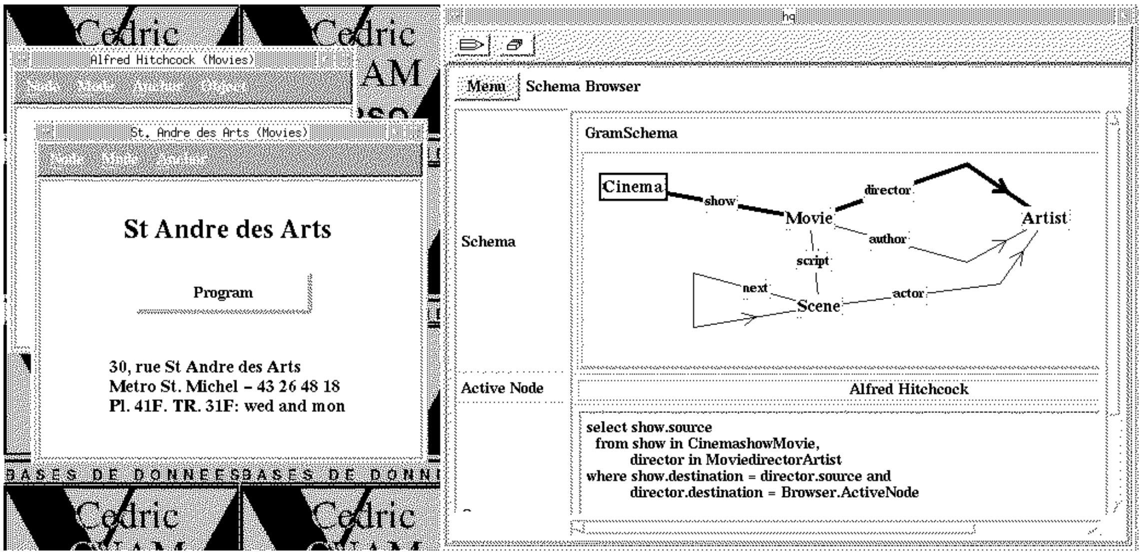


Figure 2: All cinemas featuring a movie of Alfred Hitchcock

*The bottom window of the schema browser shows the automatic translation of the equivalent Gram query into $O_2$SQL, the query language of $O_2$. The destination documents of these paths have to be the active document Alfred Hitchcock. Finally, the answer to this query, i.e. the document* `St. André des Arts`*, is displayed on to the screen.*

**Example 4.2** *The definition of a virtual (reverse) link by using the schema browser is illustrated in Figure 3. In order to define a virtual link from the document* `Twin Peaks` *to the cinemas featuring this movie, the author has created a button* [*Cinemas*] *and attached to it a script (displayed by the script editor) that includes the query generated by the selection of the link type* `show` *in the hypertext schema.*

Such an *intensional browsing* (Batini, Catarci, Costabile, and Levialdi 1991) reconciles hypertext browsing with querying in several ways :

- The user is not obliged to learn any formal query language, but keeps the same interaction style, both for following hypertext links and virtual links.
- The conceptual schema of the application is displayed on the screen and can be manipulated under a uniform interface.
- In order to define virtual links, the queries generated by the schema browser can automatically be attached to buttons inside the hyperdocument.

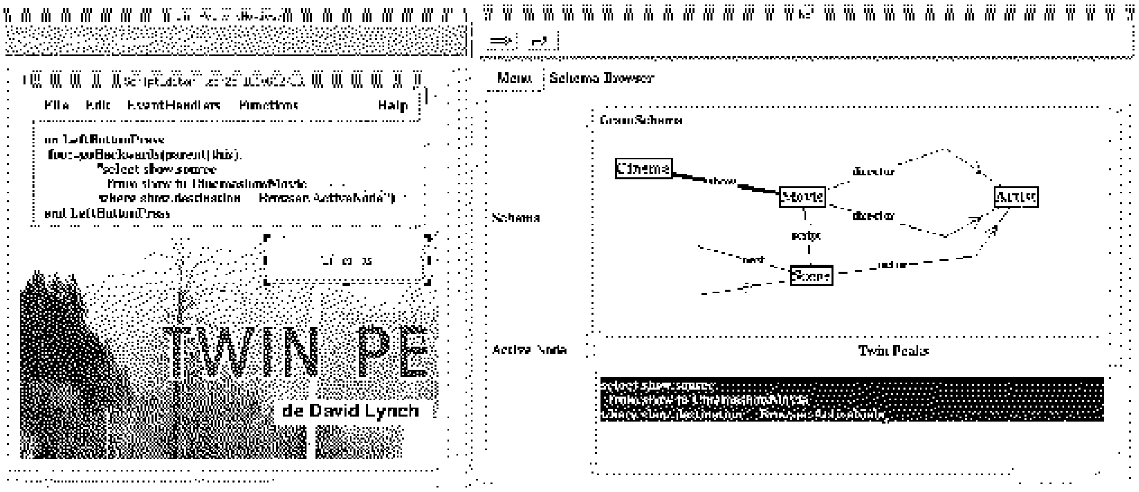


Figure 3: Defining a reverse link from a movie to its cinemas

The current implementation of this schema browser is limited to the description of simple path expressions. Whereas this is sufficient to demonstrate the benefits of such a tool, we plan to extend it to a complete visual query interface supporting the definition of joins, selections, projections and Kleene closure.

# 5 Multiple-Schema Hypertext Organization

The mechanisms we have described so far assumed that a hypertext is an instance of a single predefined schema. However, there are many situations where a single schema is either not desirable or not possible to achieve. Such situations are generated by :

- requirements for hypertext links between already available heterogeneous information based on an information network. This is typically illustrated by the WWW project (Berners-Lee, Cailliau, Groff, and Pollermann 1992) and its need for better information organization. In the WWW scenario, local servers consist very often of highly typed and structured databases which are not exploited at their real value by the loose connection the WWW provides.
- requirements for a better organization of a single hypertext base through decomposition and association of a number of multiple-schema hypertexts. These situations typically arise when the semantics of the application naturally gives rise to a hierarchical decomposition into many sub-hypertexts, or when one hypertext application might want to create/update its own hypertext which reuses or shares information stored in some already existing and separately maintained hypertext. As an example of hierarchical organization, consider passenger transportation networks. A high-level hypertext might represent cities flight interconnection (one document node per city), while with each city there might be associated a low level network of local station-airport connections by car, train and bus.

Organization of complex hypertexts is done traditionally via the well known composition mechanism at instance level. Composite nodes, contexts, or groups of nodes serve as a hierarchical backbone to the hypertext. This method however has its shortcomings :

- The hierarchy is managed at instance level and does not take into account the various information types and schemas.
- Cross-reference links can traverse the hierarchy regardless of the organization semantics.
- Hierarchical links are another type of link that requires an orthogonal management to the cross-reference link type both at user and system level.
- Hierarchy does not cater for hypertext reuse and sharing of information, especially between multiple-schema based hypertexts.

We suggest below a structuring mechanism that organizes hypertext design at type and schema level. The result is better management of encapsulation of hypertexts as well as federation of existing hypertexts, whilst simplifying the author's view of the hypertext by getting rid of the artificial hierarchical structure at instance level. We illustrate our mechanism through an example in which a CULTURAL PROGRAM hypertext (schema of Figure 4) is coupled to the already existing MOVIE GUIDE hypertext.

Our structuring mechanism allows to connect a node $n$ from a given hypertext $H$ to another hypertext $H'(n)$. We assume of course that designers of separate hypertexts conceptualize domains in compatible ways. Of course, this might be a restrictive assumption in various real-life situations. More specifically, we assume the following simple scheme : in some nodes $n$ of $H$, the user has the choice between (1) following a link in $H$ and (2) entering (zooming in) another hypertext $H'(n)$, following a path in $H'$ (entering another hypertext $H''$, etc.) and eventually come back to $n$ in $H$. Using this mechanism, it becomes possible to define queries involving paths in different loosely-coupled hypertexts.

The following examples illustrate the above scheme as well as the usage of queries in the context of loosely-coupled typed hypertexts. Figure 4 shows two hypertext schemas. In addition to the already presented MOVIE GUIDE schema (Section 2), we can see a second schema CULTURAL PROGRAM describing cultural programs for tourists.

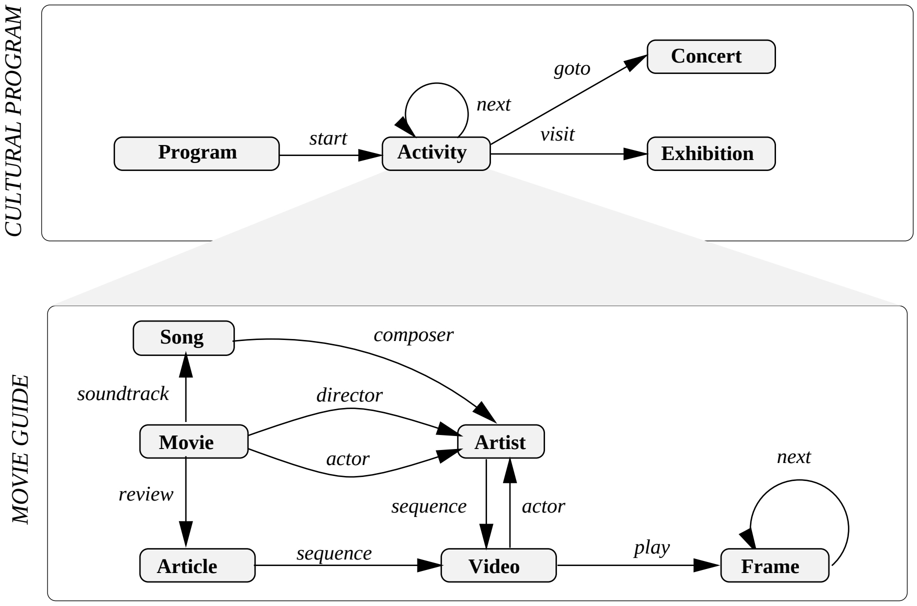


Figure 4: Cultural Program and Movies

A program is composed of different activities each of which is described in a document of type `Activity` and represents one stop in a guided tour defined by links of type `next`. The schema distinguishes three types of activities : (1) going to the cinema, (2) visiting an exhibition or (3) going to a concert. Looking at this schema, we can think of two independent hypertext applications. Nevertheless, readers of the CULTURAL PROGRAM hypertext might want to enter or "zoom into" the MOVIE GUIDE hypertext in order to obtain more information about the featured movies.

**Definition 5.1** *Let $S$ be a hypertext schema and $I(S)$ denote the set of hypertext instances of $S$. Let $t$ be a node type (not necessarily in $S$) and $I(t)$ denote the set of node instances of type $t$. A* zoom *from $t$ into $S$, denoted $zoom : I(t) \to I(S)$, is a function that associates with each node of type $t$ a (possibly empty) hypertext with schema $S$.*

By this definition, we can define $zoom : I(Activity) \to I(Movie\ Guide)$ from nodes of type `Activity` into MOVIE GUIDE hypertexts. Each document of type `Activity` now contains a button $\boxed{\Downarrow}$ that can be activated in order to enter the MOVIE GUIDE hypertext. Clicking on button $\boxed{\Downarrow}$ in activity $\mathcal{A}$ suspends navigation in the CULTURAL PROGRAM hypertext, activates the zoom and opens the hypertext $zoom(\mathcal{A})$ which describes the movie associated with $\mathcal{A}$. Afterwards, in order to resume navigation in the CULTURAL PROGRAM hypertext, a "return" button $\boxed{\Uparrow}$ has to be clicked on which brings the user back to $\mathcal{A}$.

Instead of following a path from $\mathcal{A}$ into $zoom(\mathcal{A})$ and navigate, say from an activity to the featured movie, we might directly apply a query $Q$ on $zoom(\mathcal{A})$. The activation of the zooming button $\boxed{\Downarrow, Q}$ then enters hypertext $H = zoom(\mathcal{A})$ and applies query $Q$ on $H$. Furthermore, a query might involve both, a path $p$ on activities in the CULTURAL PROGRAM hypertext as well as a path $p'$ on movies and artists embedded in $p$.

**Definition 5.2** *Let $Q$ be a set of paths satisfying some regular expression $r$. Let $t$ be a type in $r$ and $zoom : I(t) \to I(S)$ be a zooming function. Let $Q'(H)$ be a query on a hypertext instance $H$ of schema $S$. A path*

*$p' \in Q'(H)$ is embedded in a path $p \in Q$ if $p$ contains a document $d$ of type $t$ ($\tau(d) = t$) such that $p'$ is a path in $H = zoom(d)$ :*

$$embed(Q, Q') = \{(p, p') \quad | \quad p \in Q \wedge \exists d \in p : \tau(d) = t \wedge p' \in Q'(zoom(d))\}$$

*defines all pairs of paths $(p, p')$ such that $p'$ can be followed from path $p$. Let $(p, p') \in embed(Q, Q')$ be a pair of embedded paths such that $p$ satisfies $r = utv$ and $p'$ satisfies $r'$. Then $(p, p')$ is said to satisfy the regular expression $s = ut.(r')v$.*

The above definition selects paths $p$ in $H$, and for each path $p$ going through document $d$, a set of paths $p'$ in $zoom(d)$. There might be one or several entry points in $zoom(d)$. Specifying the entry points, implies restricting the path expressions $r'$ that $p'$ must satisfy.

**Example 5.1** *The following query gets the cultural programs with at least one activity where it is possible to see a film directed by Robert Altman :*

```
SELECT Program
  FROM Program start (Activity next)* Activity.Movie director Artist
 WHERE Artist.name = "Robert Altman"
```

*The dot notation is used for accessing attribute values (`Artist.name`) as well as paths in the "zoomed" hypertexts (`Activity.Movie director Artist`).*

**Example 5.2** *The following example selects all programs (guided tours) where the first activity is to see a movie with Romy Schneider followed by a visit of an exhibition in the Louvre museum.*

```
SELECT *
  FROM Program start Activity.(Movie actor Artist) next Activity visit Exhibition
 WHERE Artist.name = "Romy Schneider"
   AND Exhibition.address = "Le Louvre"
```

# 6 Implementation

Some of the above mechanisms have been implemented with the Gram model and validated on top of the hypermedia system Multicard (Rizk and Sauter 1992) and the object-oriented DBMS $O_2$ (Deux 1989). The overall system architecture is shown in Figure 5.

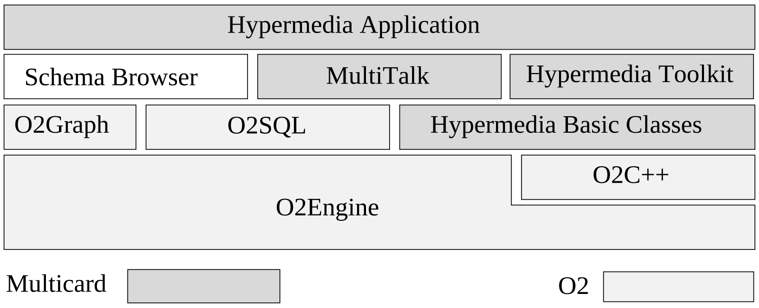


Figure 5: Multicard / $O_2$ Architecture

In a first step, we have connected Multicard with $O_2$ by using the C++ programming interface of $O_2$ (Amann, Christophides, and Scholl 1993). This integration provides a clean separation of the hypertext structure (storage layer) stored in an $O_2$ database from the node contents (within-component layer) manipulated by Multicard compliant editors (MCEditor, Emacs, Go, etc.).

Afterwards, in order to introduce new hypertext query facilities, we have extended the Multicard data model with typed nodes and links. We used $O_2$SQL (Bancilhon, Cluet, and Delobel 1989), the query language of $O_2$, which is powerful enough to express Gram queries without Kleene closure, and concentrated our attention to the implementation of a schema browser on top of $O_2$ (Section 4). $O_2$SQL can be used as an ad-hoc query language, but also be embedded into code written in the $O_2$C programming language. An important generic feature of the $O_2$SQL language is the possibility to construct new values from existing objects and values. In our context, this feature is especially useful for creating paths between nodes in the form of lists of link objects.

# 7 Conclusions

The general problem of creating and managing large-scale hypertext applications as well as simplifying navigation in large hypertexts remains one of the key issues for the next generation of hypertext systems. Previous research efforts have gone a long way into resolving this issue. Such efforts vary widely depending on the approach adopted. Some propose tailored design methodologies in the form of a set of do's and don'ts for effective authoring. Others propose automatic creation of hypertext links as an aid to authoring coupled to information retrieval techniques for navigation.

Our work builds on the approach that consists in using strong typing as a means of constraining the author's choices as well as for developing query-assisted navigation. Although this approach is considered controversial by some, our choice results from the positive conclusions of previous research which has been carried out along two largely independent main lines, namely :

- typing as a good management model for authoring. This has been applied in pretyped hypertext systems such as IBIS structure based applications (SEPIA and gIBIS), as well as in schema-based authoring (such as HDM) and strongly typed systems such as MacWeb.
- querying in hypertext is an effective user tool, complementary to navigation. We have seen this in work by (Consens and Mendelzon 1989; Afrati and Koutras 1990; Beeri and Kornatzky 1990; Amann and Scholl 1992; Lucarella et al. 1993).

Our approach has both of the above advantages. In addition, we have demonstrated how schemas and queries in typed hypertexts can be used not only as a user tool, but also as an authoring/design mechanism. In particular, we have shown how queries in our model could be used by a designer as a generic means for building more tailored authoring/navigation tools, and we have illustrated this concept through the classical guided tours and virtual links examples.

The zooming function we introduced allows a new information structuring through interconnected hypertexts. The above mechanisms can be applied, beyond a single typed hypertext, to such interconnected hypertexts each having its own schema and types.

Most of these mechanisms have been implemented on top of the hypermedia system Multicard/$O_2$. An important feature of this prototype is a schema browser that lets the user formulate queries during hypertext navigation. We plan to extend this graphical query interface to cater for all operations of the Gram algebra such as join, concatenation, projection and selection and Kleene closure.


### Acknowledgments

This work was supported in part by the French *Programme de Recherche Coordonnée BD3* and the BRA Esprit Project *Amusing*. The authors would like to thank the anonymous reviewers whose careful reading and comments helped in improving the quality of the paper.


# References


Afrati, F. and C. Koutras (1990, November). A hypertext model supporting query mechanisms. In *Proceedings of the First European Conference on Hypertext and Hypermedia (ECHT'90)*, Versailles, pp. 52–66.

Amann, B. (1994, February). *Interrogation d'Hypertextes.* Ph. D. thesis, Conservatoire National des Arts et Métiers, Paris, France.

Amann, B., V. Christophides, and M. Scholl (1993, September). HyperPATH/O2: Integrating hypermedia systems with object-oriented database systems. In *Proceedings of the Fourth International Conference on Data and Expert Systems Applications (DEXA93)*, Prague, Czech Republic.

Amann, B. and M. Scholl (1992, December). Gram: A graph data model and query language. In *Proceedings of the Fourth ACM Conference on Hypertext and Hypermedia (ECHT'92)*, Milano, Italy.

Apple Computer Inc. (1987). *MacIntosh HyperCard User's Guide.* CA, USA: Cupertino.

Bancilhon, F., S. Cluet, and C. Delobel (1989). A query language for the $O_2$ object-oriented database system. In *Proceedings of the Second International Workshop on Database Programming Languages (DBPL'89).*

Batini, C., T. Catarci, M. Costabile, and S. Levialdi (1991). Visual strategies for querying databases. In *IEEE Workshop on Visual Languages*, pp. 183–189.

BEERI, C. AND Y. KORNATZKY (1990, November). A logical query language for hypertext systems. In *Proceedings of the First European Conference on Hypertext and Hypermedia (ECHT'90)*, Versailles, pp. 67–80.

BERNERS-LEE, T., R. CAILLIAU, J. GROFF, AND B. POLLERMANN (1992). World-Wide Web: The information universe. *Electronic Networking: Research, Applications and Policy 2*(1), 52–58. Meckler Publishing, CT, USA.

BIEBER, M. AND J. WANG (1994, September). Backtracking in a multiple-window hypertext environment. In *Proceedings of the ACM European Conference on Hypermedia Technology (ECHT'94)*, Edinburgh, Scotland, pp. 158–166.

CHEN, J., T. EKBERG, AND C. THOMPSON (1990). Querying an object-oriented hypermedia system. In R. MacAleese and C. Green (Eds.), *Hypertext : State of the Art*, Chapter 25, pp. 231–238. Oxford, England: Intellect.

CONKLIN, J. AND M. BEGEMAN (1988, October). gIBIS: a hypertext tool for exploratory policy discussion. *ACM Transactions on Information Systems 6*(4), 303–331.

CONSENS, M. AND A. MENDELZON (1989, November). Expressing structural hypertext queries in GraphLog. In *Proceedings of the Hypertext'89 Conference*, Pittsburgh, Pennsylvania, pp. 269–292.

CONSENS, M. AND A. MENDELZON (1990). GraphLog: a visual fromalism for real life recursion. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Nashville, Tennessee, pp. 404–416.

COOMBS, J. (1990, September). Hypertext, full text and automatic linking. In *Proceedings of the 13th International SIGIR Conference on Research and Development in Information Retrieval*, pp. 83–98.

CROFT, W. AND H. TURTLE (1989, November). A retrieval model for incorporating hypertext links. In *Proceedings of the Hypertext'89 Conference*, Pittsburgh, Pennsylvania, pp. 213–224.

DEROSE, S. (1989, November). Expanding the notion of links. In *Proceedings of the Hypertext'89 Conference*, Pittsburgh, Pennsylvania, pp. 249–257.

DEUX, O. (1989, March). The Story of $O_2$. *IEEE Transactions on Knowledge and Data Engineering 2*(1), 91–108.

DUNLOP, M. AND C. RIJSBERGEN (1991, April). Hypermedia and probabilistic retrieval. In *Intelligent Text and Image Handling (RIAO '91)*, Barcelona, Spain.

FORD, S., J. JOSEPH, D. LANGWORTHY, D. LIVELY, G. PATHAK, E. PEREZ, R. PETERSON, D. SPARACIN, S. THATTE, D. WELLS, AND S. AGARWAL (1988). Zeitgeist : Database support for object-oriented programming. In K. Dittrich (Ed.), *Advances in Object Oriented Database Systems*, pp. 23–42. Berlin: Springer Verlag.

FRISSE, M. (1988, July). Searching for information in a hypertext medical handbook. *Communications of the ACM 31*(7), 880–886.

GALLAGHER, L., R. FURUTA, AND P. STOTTS (1990). Increasing the power of hypertext search with relational queries. *Hypermedia 2*(1), 1–14.

GARZOTTO, F., L. MAINETTI, AND P. PAOLINI (1993, January). Navigation patterns in hypermedia databases. In *Proc. of the $26^{th}$ Hawai International Conference on System Science*, Maui, pp. 370–379.

GARZOTTO, F., P. PAOLINI, AND D. SCHWABE (1993, January). HDM – a model-based approach to hypertext application design. *ACM Transactions on Information Systems 11*(1), 1–26.

GRUDIN, J. (1994, January). Groupware and social dynamics : Eight challenges for developers. *Communications of the ACM 37*(1), 93–105.

HALASZ, F. (1988, July). Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM 31*(7), 836–852.

LUCARELLA, D., S. PARISOTTO, AND A. ZANZI (1993, November). MORE: Multimedia object retrieval environment. In *Proceedings of the Fifth ACM Conference on Hypertext (Hypertext'93)*, Seattle, Washington, pp. 39–50.

MARCHIONINI, G. AND B. SHNEIDERMAN (1988). Finding facts vs. browsing knowledge in hypertext systems. *IEEE Computer 19*(1), 70–80.

MARSHALL, C., F. HALASZ, R. ROGERS, AND W. JANSSEN (1991, December). Aquanet: a hypertext tool to hold your knowledge in place. In *Proceedings of the Third ACM Conference on Hypertext (Hypertext'91)*, San Antonio, Texas, pp. 261–275.

MARSHALL, C. AND P. IRISH (1989, November). Guided tours and on-line presentations: How authors make existing hypertext intelligible for readers. In *Proceedings of the Hypertext'89 Conference*, Pittsburgh, Pennsylvania, pp. 15–26.

McCall, R., P. Bennett, P. D'Oronzo, J. Ostwald, F. Shipman, and N. Wallace (1990). PHIDIAS : Integrating cad graphics into dynamic hypertext. In *Proceedings of the First European Conference on Hypertext and Hypermedia (ECHT'90)*, Versailles, France, pp. 152–165.

McCall, R., I. Mistrik, and W. Schuler (1980, October). An integrated information and communication system for problem solving. In P. Glaeser (Ed.), *Proceedings of the Seventh International CODATA Conference*, Kyoto, Japan. Pergamon Press.

Nanard, J. and M. Nanard (1991, December). Using structured types to incorporate knowledge in hypertext. In *Proceedings of the Third ACM Conference on Hypertext (Hypertext'91)*, San Antonio, Texas, pp. 329–343.

ODMG (1994). *The Object Database Standard : ODMG-93 : Release 1.1.* Morgan Kaufmann.

Osterbye, K. and K. Normark (1994, September). An interaction engine for rich hypertext. In *Proceedings of the ACM European Conference on Hypermedia Technology (ECHT'94)*, Edinburgh, Scotland, pp. 167–176.

Rada, R. (1991). Small, medium and large hypertext. *Information Processing & Management 27*(6), 659–677.

Rizk, A. and L. Sauter (1992, December). Multicard: An open hypermedia system. In *Proceedings of the Fourth ACM Conference on Hypertext and Hypermedia (ECHT'92).*

Rossiter, B., T. Sillitoe, and M. Heather (1990, August). Database support for very large hypertexts. *Electronic Publishing 3*(3), 141–154.

Streitz, N., J. Hannemann, and M. Thüring (1989). From ideas and arguments to hyperdocuments : Traveling through activity spaces. In *Proceedings of the Hypertext'89 Conference*, pp. 343–364.

Trigg, R. (1988, October). Guided tours and tabletops: Tools for communicating in a hypertext environment. *ACM Transactions on Information Systems 6*(4), 398–414.

Utting, K. and N. Yankelovich (1989). Context and orientation in hypermedia networks. *ACM Transactions on Information Systems 7*(1), 58–84.

Zellweger, P. (1989, November). Scripted documents: A hypermedia path mechanism. In *Proceedings of the Hypertext'89 Conference*, Pittsburgh, Pennsylvania.