Querying Typed Hypertexts in Multicard/O_2[*]

Bernd Amann, Michel Scholl
Cedric CNAM, 292 Rue St. Martin, 75141 Paris Cedex 03 France
INRIA, Rocquencourt, 78153 Le Chesnay Cedex France
e-mail: {amann,scholl}@cnam.fr

Antoine Rizk
Euroclid
12 Avenue des Près
78180 Montigny le Bretonneux France
e-mail: Antoine.Rizk@inria.fr

ABSTRACT

Due to the growing complexity of modern hypertext applications, current hypertext systems require new mechanisms to support authoring and user navigation through large sets of documents connected by links. A general solution is to extend hypertext systems to cater for semantics of application domains. This requires new hypertext models providing strongly typed documents and links. Such models have been proposed and put to use in systems such as HDM and MacWeb to facilitate authoring of large hypertexts. In addition, Gram and MORE use typing and graph-based hypertext schemas for querying hyperdocuments. In this paper, we will show how query languages could be further exploited for designing sophisticated general query-based navigation mechanisms. We illustrate our examples using the Gram model and describe an implementation with the hypermedia system Multicard connected to the object-oriented database management system O2.

KEYWORDS: hypertext querying, browsing, hypertext schema, visual interface

1 INTRODUCTION

Hypertext systems are generally viewed as an example of graph structured information systems based on navigation. (Multimedia) data is stored in nodes or documents which are connected by links. Browsing is the typical way of accessing data in hypertext systems. In contrast with Database Management Systems (DBMS) which are based on data models with strong typing (relations, classes), hypertext documents have so far been viewed as weakly typed [18, 28], and users not only traversed hypertext links independently of their type, but also could add new documents and links without necessarily specifying their semantics in form of a type.

It has now become clear that the above distinction between hypertext systems and DBMS is oversimplified and no longer valid. A first step towards "typed" hypertexts were formal models [24, 15] based on the semantic data model IFO [1]. Subsequent implementations like HDM [16] and MacWeb [22] have then demonstrated that the best way for managing "real-size" hypertexts is by generating them according to a schema describing the hypertext application. Hypertext schemas have therefore been put to complementary use in authoring hypertext applications. Other data models, such as Gram [4] and MORE [20] have moreover exploited schemas for querying the resulting hyperdocuments in a more structured way.

The use of graphs for describing the structure of information as well as for formulating complex queries in large applications is not new. Semantic data models such as the Entity-Relationship model [7] and IFO [1] are examples of paradigms which make intensive use of graph based diagrams, useful not only for representing database schemas, but also as a help for navigating through data. Although database query languages such as SQL [29] are declarative, a complex query including a large number of relations, say more than 3, is more naturally conceptualized and prepared by navigating among relations. Visual query interfaces [6] based on graph representations of the schema, then, become extremely useful. This is even more obvious in object-oriented databases. With object-oriented query languages such as O2SQL [5] or XSQL [19] users traverse composition links between classes by means of path expressions. The application of path expressions for querying structured (SGML) documents without exact knowledge of their schema is studied in [8]. MORE [20], GraphLog [9] and GOOD [17] are also based on a graph representation of

[*]This article appeared in "ECHT'94, Edinburgh, Scotland, September 1994

data, but in contrast with other query languages, queries are graph patterns matched against subgraphs of the database.

Querying in this way allows us to define a variety of navigation strategies. The objective of this paper is to investigate some query-based browsing mechanisms for implementing such strategies. While we shall use the Gram path algebra [4] for the presentation of these mechanisms, they should also be applicable on top of other data models such as MORE, GraphLog or any relational or object-oriented data model.

We first introduce Gram through examples of data and queries in Section 2. For a complete formal presentation of Gram, see [4]. Section 3 is devoted to the application of query languages to hypertext navigation. We introduce a few basic mechanisms, namely virtual link, reverse link and user history. Section 4 introduces a better information structuring through the definition of a zooming function for connecting several hypertexts. Zooming allows us in particular reutilization of existing hypertexts. It is also useful for modeling hierarchical relationships. The implementation of several of these mechanisms on top of the hypermedia system Multicard [25] and the object-oriented DBMS O2 [13] is described in Section 5.

2 THE GRAM MODEL

A Gram schema is a directed labeled graph where nodes represent document types and edges correspond to the possible types of links between documents. Figure 1 shows the schema of a hypertext storing multimedia information about cinemas, movies and artists.

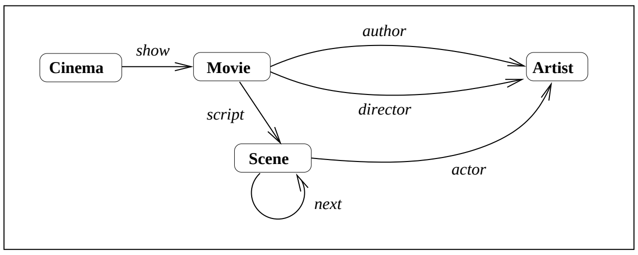


Figure 1: A Movie Guide Schema

A document can have one of the types `Cinema`, `Movie`, `Artist` and `Scene`. Cinemas (documents of type `Cinema`) are connected to the featured movies (documents of type `Movie`) by links of type `show`. Each movie is connected to its director and its authors by links of type `director` and `author` respectively. A movie's script is a sequence of scenes connected by links of type `next`. Finally, one may obtain information about a movie's actors by following links of type `actor` starting from its `Scene` documents. Note that links may contain information such as a room number (type `show`) or an actors role (type `actor`).

The Gram query language is based on a path algebra and makes it possible to

- select paths according to regular expressions over document and link types,
- select paths according to document and link values,
- construct new paths by projection and concatenation,
- join paths sharing document.

For illustration, assume a user is interested in all cinemas featuring a movie directed by Alfred Hitchcock (an SQL like syntax is used for clarity):

```
SELECT Cinema
  FROM Cinema show Movie director Artist
 WHERE Artist.name = ''Alfred Hitchcock''
```

Paths satisfying the regular expression `Cinema show Movie director Artist` are selected (`FROM`-clause) and restricted to the `WHERE`-clause condition.

The answer to the following query contains all authors of movies directed by Ingmar Bergman. Note that the paths in the range of the query (`FROM`-clause) have to share the document of type `Movie`, and authors are distinguished from directors by renaming one occurrence of type `Artist` into `Artist_2`:

```
SELECT Artist
  FROM Movie author Artist+Movie director Artis
 WHERE Artist_2.name = ''Ingmar Bergman''
```

The hypertext schema shown in Figure 1 is cyclic (edge type next). This allows the application of Kleene closure, as it is illustrated in the next query that gets all movies with Romy Schneider:

```
SELECT Movie
  FROM Movie script (Scene next)*Scene
            actor Artist
WHERE Artist.name = ''Romy Schneider''
```

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) since it is supposed [2] that hypertexts are *connected* graphs. However Gram allows us to keep the whole information about paths and not only about their extremities.

3 QUERY-BASED NAVIGATION

This section is devoted to the application of query languages to user navigation in hyperdocuments. We shall describe several query-based browsing mechanisms for implementing high-level hypertext navigation strategies.

3.1 Virtual links

Existing hypertext systems such as the Dynamic Medical Handbook [14], I^3R [11], Intermedia [10] or MacWeb [23] support advanced information retrieval and indexing mechanisms for the definition of virtual or intensional links [12]. Opposed to extensional "hard-wired" links that are defined by two extremities, virtual links are the result of queries on the hyperdocument contents and structure. Independently of the underlying data model, we suppose that all queries return sets of paths in some hypertext network.

A virtual link defined by a (path) query Q and denoted by link(Q), is the set of node (document) pairs connected by paths in Q:

link(Q) = $\{(d_1, d_k) \mid d_1 l_1 ... l_{k-1}\ d_k \in Q\}$.

Any such documents d_1 and d_k are said to be connected by a virtual link ∈ link(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 d by the operation goto:

goto(d, Q) = {d' | (d, d') ∈ link(Q)},

i.e. goto(d, Q) displays all destination documents of paths contained in Q and starting in document d. In the following, document d will be called the *active document*.

Instead of opening all documents related to d simultaneously, some control strategy has to be used to avoid an "explosion" of the number of windows displayed on the screen: if the number of documents found is less than some limit, say 4, all documents are opened and displayed on the screen simultaneously. Otherwise, the user gets some menu where he can choose the documents to be opened.

Virtual links introduce flexibility and allow the control of the hypertext size by avoiding unnecessary extensional links between documents. Note also that (1) goto(d, Q) might be precomputed for faster navigation - which implies recalculation when the hypertext graph is modified - and (2) this mechanism could be implemented in a relational model by a view mechanism.

Example 3.1 In order to obtain the actors playing in the movie Jurassic Park, a user might read the scenario by following links of type `next` and display documents about actors by traversing links of type `actor`. A better solution is to use the following query for traversing the virtual link {(Jurassic Park, Jeff Goldblum), (Jurassic Park, Richard Attenborough), (Jurassic Park, Greg Burnson), ...}:

```
SELECT Artist
  FROM Movie script(Scene next)*Scene
            actor Artist
 WHERE Movie.title = ''Jurassic Park''
```

Instead of selecting the document Jurassic Park by the condition `Movie.title = ''Jurassic Park''`, one might also define a "generic" query, where the formal parameter `Active Document` is replaced by the active document (here Jurassic Park):

```
SELECT Artist
  FROM Movie script(Scene next)*Scene
            actor Artist
 WHERE Movie = Active Document
```

Afterwards, in order to display all actors playing in a different movie, the user only has to activate the corresponding document before evaluating the same query.

This example also illustrates how virtual links prevent unnecessary navigation by following several paths to the same document: although there exist different scenes with the same actor, the latter appears only once in the result of the query.

Hypertext browsing implies an asymmetric view of data: 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:

comingfrom(d, Q) = {d' | (d', d) ∈ link(Q)}

3.2 Schema browsing

A virtual link goto(d,Q) can be attached to a button $\boxed{\rightarrow,\mathbf{Q}}$ in some document d. Clicking on this button then triggers the interpretation of Q and opens the documents in goto(d, Q). Another solution is to provide 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 interactively select node and link types in order to create path expressions[1]

Example 3.2 For displaying all cinemas featuring a movie directed by Alfred Hitchcock, the user activates the document Alfred Hitchcock and selects the link types `director` and `show` in the schema graph representation of the movie guide (Figure 1). This selection is translated automatically into the following query finding the corresponding `Cinema` documents:

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

The implementation of a schema browser on top of the O2 DBMS and the Multicard hypertext system will be described in Section 5.

[1] Observe that we use the term *schema browsing* in the sense of following link types in the graph representation of the hypertext schema.

3.3 Navigation space restriction

Navigation, even when query assisted, becomes extremely complex when the number of displayed nodes and edges passes some limit. Navigation space restriction is then necessary. Several mechanisms have been proposed [30, 26]. We suggest to restrict navigation to paths defined by a query Q. Then the user navigates step by step but only in the predefined space: in each document only buttons corresponding to the paths of Q are active, the others being disabled. As for virtual links this mechanism can be implemented either by storing Q with a button $\boxed{\otimes,\mathbf{Q}}$ or by ad-hoc querying.

3.4 Querying history

Most hypertext systems allow to return to some previously visited document by using a special “back” button $\boxed{\hookleftarrow}$. Reverse links as defined above (comingfrom) 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 previous movie Le Dernier Metro, but to all movies directed by François Truffaut.

We shall implement user history as follows. Each time a link is followed, the source document (active document) is pushed onto a stack H. Upon the display of a document, thanks to the above stack mechanism, the user can navigate back to the previous document (pop(H)) or to the first one (empty(H)). Combined with some query language it is then possible to implement more sophisticated reverse strategies: instead of following the history step by step (button $\boxed{\hookleftarrow}$)), a whole path p returned by a query Q might be popped from the history: let p be a path connecting documents d_1, d_2,...$d_k$in the same order as they were pushed onto H. We assume that d_k is the active document and has already been pushed on H. Then p can be popped from H and d_1 becomes the active document: pop(H, p).

As we have done for the definition of virtual links, we can use a “back” button attached to a query Q: $\boxed{\hookleftarrow,\mathbf{Q}}$. By clicking on this button, the starting documents of the paths p in Q that can be popped from H will be displayed.

Example 3.3 A user reading the script of the movie Germinal (following a path satisfying the regular expression `Movie script (Scene next)* Scene actor Artist)` has obtained the document about some actor by following a link of type actor. Clicking on $\boxed{\hookleftarrow,\textbf{Movie script (Scene next)* Scene actor Artist}}$ then allows to return immediately to the document Germinal.

4 HYPERTEXT REUTILIZATION

The following mechanism allows a better connection and structuring between separate hypertexts. In particular, it becomes possible to “call” a hypertext H from some document in another hypertext H’. Looking at the schema in Figure 2, we can think of one hypertext on cinemas and their schedules and one on movies, directors, authors and actors. Each document of type `Schedule` now contains a button $\boxed{\downarrow}$ that can be activated in order to enter the corresponding hypertext about the featured movies.

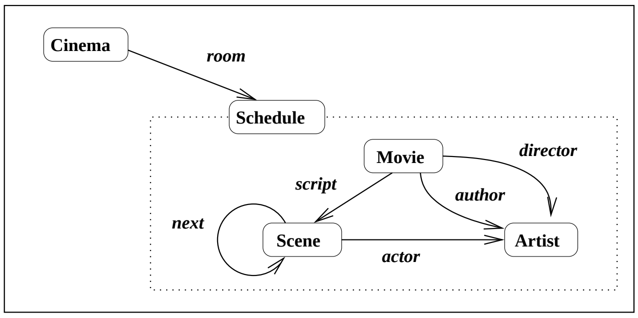


Figure 2: Cinemas and Movies

Zooming Function: Let S be a hypertext schema, t be a node type, and I(S) denote the set of hypertexts with schema S. Then zoom: I(t) → I(S) associates with a node of type t a hypertext with schema S.

Clicking on button $\boxed{\downarrow}$ in schedule s activates the zoom and opens the hypertext zoom(s). Instead of following a path from some schedule s into zoom(s) and navigate, say from a schedule to the directors of the movies, we might directly apply a query on zoom(s): the activation of the zooming button $\boxed{\downarrow,\mathbf{Q}}$ enters hypertext H = zoom(s) and applies query Q on H.

However, a query might involve both, a path p on cinemas and schedules as well as a path p’ on movies and artists embedded in p.

Embedded Paths: Let Q be a set of paths satisfying some regular expression r. Let t be a type in r and zoom: I(t) → I(S) be a zooming function. Let Q’ be a query on hypertexts with schema S. Then we can enter a path p’∈Q’ from a path p’∈Q if p contains a document d such that p’ is a path in zoom(d):

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

defines all pairs of paths (p, p’) such that p’ can be entered from path p.

For representing such pairs of paths, we will use the following rewriting rule where a dot notation is used to indicate the document to be zoomed:

Rewriting: Let (p, p')∈embed(Q, Q') be a pair of paths such that p satisfies r = u t v and p' satisfies r'. Then (p, p') is said to satisfy the regular expression s = ut.(r')v.

Example 4.1 The following query gets all cinemas showing films with the participation of Woody Allen (as actor, director, or author). Observe that we do not specify the paths leading to this artist inside the movies hypertext:

```
SELECT Cinemas
  FROM Cinema room Schedule.Artist
 WHERE Artist.name = ‘‘Woody Allen’’
```

The dot notation is used for accessing attribute values (`Artist.name`) as well as paths in the "zoomed" hypertexts (`Cinema room Schedule.Artist`).

An alternate (O2SQL like) syntax for this query might be the following:

```
SELECT p.Cinema
  FROM p IN Cinema room Schedule
       p’ IN p.Schedule.Artist
 WHERE Artist.name = ‘‘Woody Allen’’
```

Here p is a path in the Cinemas hypertext and satisfies the regular expression `Cinema room Schedule`. The second path p' consists of one document of type `Artist` contained in the movies hypertext that can be accessed (zoomed) from the `Schedule` document in p.

5 IMPLEMENTATION

Several of the presented mechanisms (virtual links, schema browser) have been implemented and validated on top of the hypermedia system Multicard [25] and the object-oriented DBMS O2 [5]. The overall system architecture is shown in Figure 3.

In a first step, we have connected Multicard with O2 by using the C++ programming interface of O2 [3]. According to the Dexter Hypertext Reference Model [18] this integration provides a clean separation of the hypertext structure (storage layer) stored in an O2 database from the node contents (within-component layer) manipulated by Multicard compliant editors (MCEditor, Emacs, Go, ...) [25]. Afterwards, in order to introduce new hypertext query facilities, we have extended the Multicard data model with typed nodes and links (each type corresponds to an O2 class definition).

Prior to the implementation of a complete query language based on complex path expressions (such as Gram), we decided to use O2SQL for querying Multicard hypertexts and to concentrate our attention to the implementation of a schema browser on top of O2. O2SQL is a high-level query language for querying O2 databases. It can be used as an ad-hoc query language, but also be embedded into code written

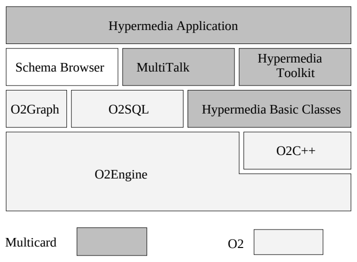


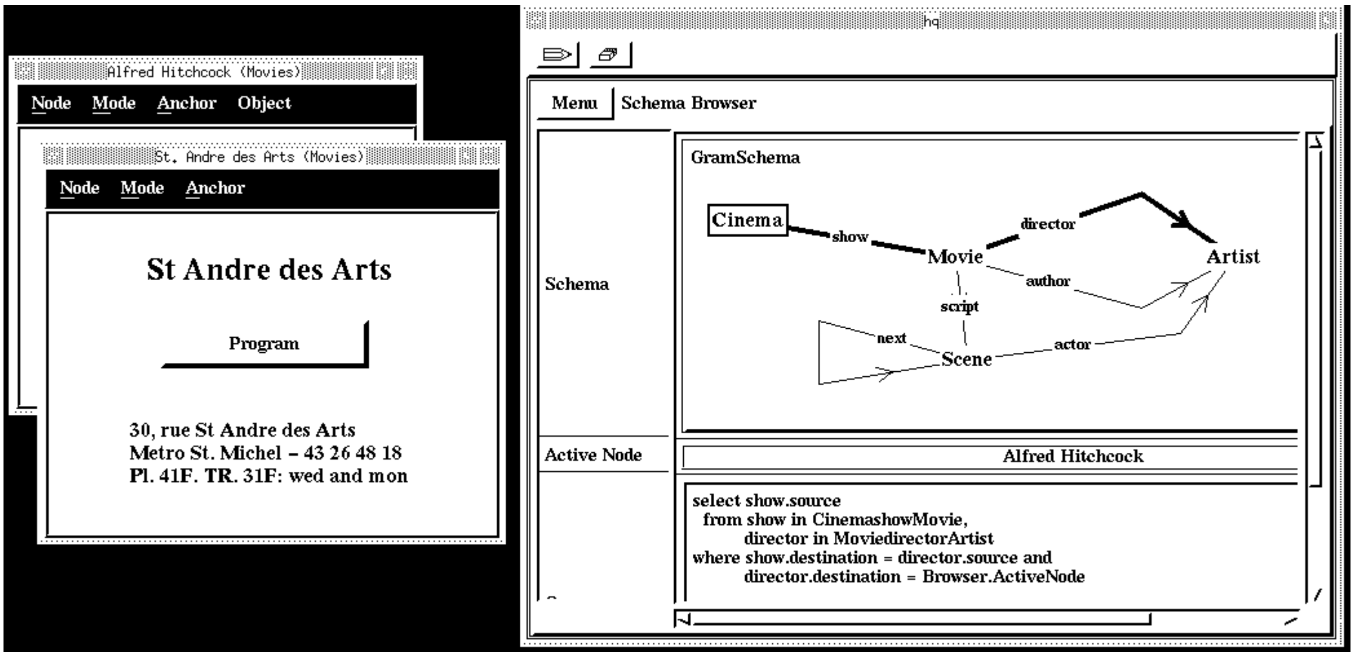
Figure 3: Multicard / O_2

in the O2C programming language. An important generic feature of the O2SQL language is the possibility to construct new values from existing objects and values. In our context, this feature is especially useful for creating paths (virtual links) between nodes in the form of lists of link objects.

Hypermedia applications are built by using the Multicard hypermedia toolkit, the Multicard script language Mul-tiTalk and the schema browser that is composed of three windows containing (1) a visual representation of the hypertext schema graph, (2) the active document (node) and (3) the generated O2SQL query that can be used ad-hoc or stored in a MultiTalk script, attached to a button in the active document.

Example 5.1 An example of an ad-hoc definition of a reverse link by using the schema browser is shown in Figure 4 (Example 3.2). 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 has then been automatically translated into an O2SQL query shown in the bottom window of the schema browser:

```
select show.source
  from show in Cinema show Movie,
       director in Movie director Artist
```



St Andre des Arts

30, rue St Andre des Arts
Metro St. Michel - 43 26 48 18
Pl. 41F. TR. 31F: wed and mon

```
select show.source
  from show in CinemashowMovie,
       director in MoviedirectorArtist
where show.destination = director.source and
       director.destination = Browser.ActiveNode
```

Figure 4: All cinemas showing a movie of Alfred Hitchcock

```
where show.destination = director.source
   and director.destination = Browser.ActiveNode
```

The from clause of the query defines two data variables show and director that range respectively over all links of type `show` and `director` (defined by the O2 classes `CinemashowMovie` and `MoviedirectorArtist`). The where clause selects those couples of links (`show`, `director`) that describe paths satisfying the regular expression `Cinema show Movie director Artist`. The destination documents of these paths have to be the active document Alfred Hitchcock. Finally, the answer to this query (`show.source`), i.e. the document St. André des Arts, is displayed on to the screen.

Example 5.2 The definition of a virtual (reverse) link by using the schema browser is illustrated in Figure 5. 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 Multicard script (displayed by the Multicard script editor) that includes the O2SQL query generated by the selection of the link type show in the hypertext schema.

Such an intensional browsing [6] reconciles hypertext browsing with querying in several ways:

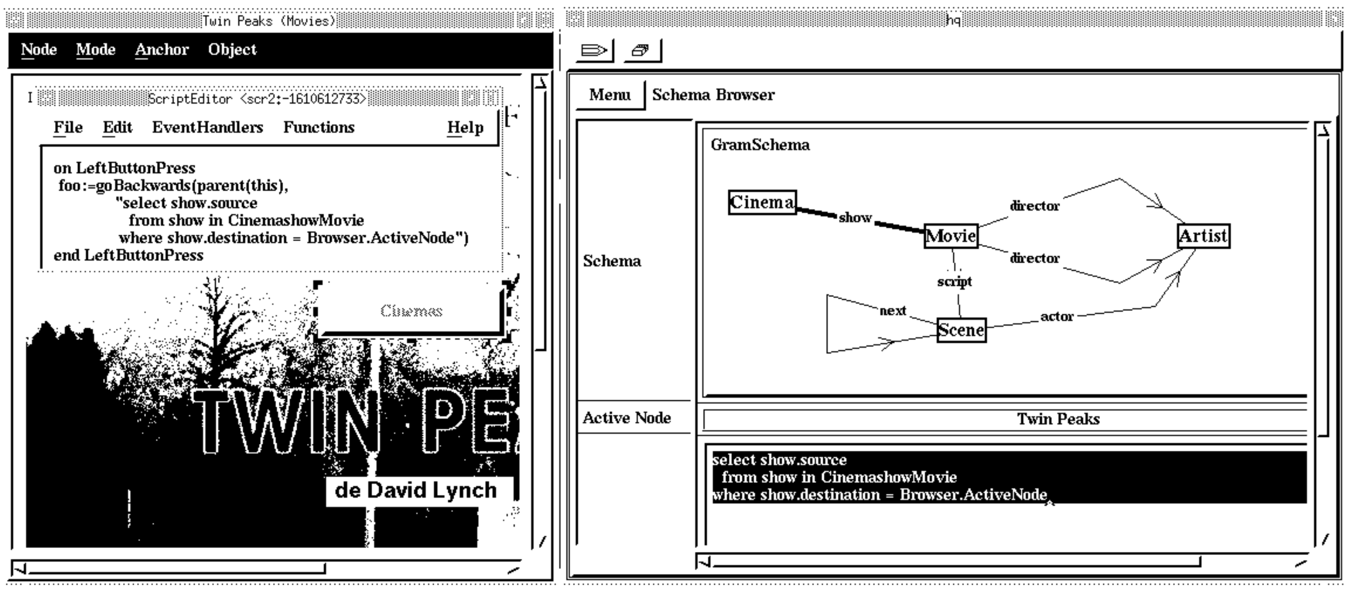
- The user is not obliged to learn any formal query language, but keeps the same interaction style, both for following hypertext links and virtual links.
- The conceptual schema of the application is displayed on the screen and can be manipulated under a uniform interface.
- In order to define virtual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

6 CONCLUSIONS

The general problem of creating large-scale hypertext applications while minimizing complexity of use remains one of the key issues for the next generation of hypertext systems. It has already been shown that the integration of hypertext systems and existing technologies for handling large volumes of data (database systems) and documents (information retrieval) results in more powerful information systems providing new facilities for representing and exploring information.

An important step in this direction was the description of explicit semantic properties by hypertext schemas [16, 22]. This extension not only allows the creation of coherent hyperdocuments in a systematic way, but also the design of new query languages and user interfaces based on graphical representations of hypertext schemas [4, 20].

This work shows how query languages for typed hypertexts can be used for implementing high-level query-based hypertext browsing. The answer of some query Q can be used as a virtual link allowing multiple-steps navigation. This mechanism introduces flexibility and allows control of the hypertext size. Similar to views in relational databases, it avoids the creation of actual edges between nodes. By restricting user navigation space to paths of a query Q, browsing in complex graphs becomes easier, especially for casual users. The well-known user history mechanism not only allows to



```
on LeftButtonPress
 foo:=goBackwards(parent(this),
          "select show.source
            from show in CinemashowMovie
          where show.destination = Browser.ActiveNode")
end LeftButtonPress
```

```
select show.source
  from show in CinemashowMovie
where show.destination = Browser.ActiveNode
```

Figure 5: Defining a reverse link from a movie to its cinemas

return to previous documents step by step, but also to create virtual links to already visited documents.

Finally, zooming was introduced as a useful information structuring mechanism. In particular, it allows a better reutilization of existing hypertexts and the modeling of hierarchical relationships. This seems to be especially useful for querying distributed hypertexts like the World Wide Web (WWW) [27] where users organize and access information without concern of its distribution (hypertext links connect documents on different hosts connected by the Internet network). Similar to index servers like WAIS [21], we might suppose the creation of schema servers that might be accessed for querying locally stored hypertexts. By using the presented zooming mechanism users might, then, not only query the contents of the documents managed by a server, but also navigate by using embedded queries.

Most of these mechanisms have been implemented on top of the hypermedia system Multicard/O2. An important feature of this prototype is a schema browser for ad-hoc querying during hypertext navigation. We plan to extend the graphical query interface and to define all operations of the Gram algebra (join, concatenation, projection, selection) by a graphical query language.

7 ACKNOWLEDGMENTS

We wish to thank V. Christophides and A.M. Vercoustre, who helped improving the quality of this paper. Thanks are also due to the Bull company, and in particular to M. Ahedo, L. Sauter, P. Dagand and F. Thorel.

References

[1] S. Abiteboul and R. Hull. IFO: A formal semantic database model. ACM Trans. on Database Systems, 12(4):525-565, December 1987.

[2] B. Amann. Interrogation d'Hypertextes. PhD thesis, Conservatoire National des Arts et Métiers, Paris, France, February 1994.

[3] B. Amann, V. Christophides, and M. Scholl. HyperPATH/O2: Integrating hypermedia systems with object-oriented database systems. In Proc. of the 4th Int. Conf. on Data and Expert Systems Applications (DEXA93), Prague, Czech Republic, September 1993.

[4] B. Amann and M. Scholl. Gram: A graph data model and query language. In Proc. of the 4th ACM Conf. on Hypertext and Hypermedia (ECHT'92), Milano, Italy, December 1992.

[5] F. Bancilhon, S. Cluet, and C. Delobel. A query language for the O2 object-oriented database system. In Proc. of the 2nd Int. Workshop on Database Programming Languages (DBPL'89), 1989.

[6] C. Batini, T. Catarci, M.F. Costabile, and S. Levialdi. Visual strategies for querying databases. In IEEE Workshop on Visual Languages, pages 183-189, 1991.

[7] P.P. Chen. The entity-relationship model: Toward a unified view of data. ACM Trans. on Database Systems, 1(1):9-36, 1976.

[8] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In Proc. of the ACM SIGMOD Conf. on Management of Data, Minneapolis, Minnesota, May 1994.

[9] M.P. Consens and A.O. Mendelzon. GraphLog: a visual fromalism for real life recursion. In Proc. of the ACM SIGACT-SIGMOD Symp. on Principles of Database Sys-

tems, pages 404-416, Nashville, Tennessee, 1990.

[10] J.H. Coombs. Hypertext, full text and automatic linking. (SIGIR 90) Research and Development in Information Retrieval, pages 83-98, 1990.

[11] W.B. Croft and H. Turtle. A retrieval model for incorporating hypertext links. In Proc. of Hypertext'89, pages 213-224, November 1989.

[12] S.J. DeRose. Expanding the notion of links. In Proc. of Hypertext '89, pages 249-257, November 1989.

[13] O. Deux. The Story of O2. IEEE Trans. on Knowledge and Data Engineering, 2(1), March 1989.

[14] M.E. Frisse. Searching for information in a hypertext medical handbook. Communications of the ACM, 31(7):880-886, July 1988.

[15] P.K. Garg. Abstraction mechanisms in hypertext. Communications of the ACM, 31(7):862-870, July 1988.

[16] F. Garzotto, P. Paolini, and D. Schwabe. HDM - a model-based approach to hypertext application design. ACM Trans. on Information Systems, 11(1):1-26, January 1993.

[17] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object model for database end-user interfaces. In Proc. of the ACM SIGMOD Conf. on Management of Data, pages 24-33, May 1990.

[18] F.G. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. In Proc. Hypertext Standardization Workshop, NIST, pages 95-133, January 1990.

[19] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In Proc. of the ACM SIGMOD Conf. on Management of Data, pages 393-402, 1992.

[20] D. Lucarella, S. Parisotto, and A. Zanzi. MORE: Multimedia object retrieval environment. In Proc. of Hypertext'93, pages 39-50, Seattle, Washington, November 1993.

[21] P. Marshall. WAIS: The Wide Area Information Server or Anonymous What??? available by anonymous ftp:

quake.think.com/pub/wais/doc/UWO-wais-paper.ps, June 1992.

[22] J. Nanard and M. Nanard. Using structured types to incorporate knowledge in hypertext. In Proc. of Hypertext'91, pages 329-343, December 1991.

[23] J. Nanard and M. Nanard. Should anchors be typed too ? In Proc. of Hypertext '93, pages 51-62, Seattle, Washington, November 1993.

[24] G. Richard and A. Rizk. Quelques idées pour une modélisation des systèmes hypertextes. T.S.I. (Technique et Science Informatiques), 9(6):505-514, June 1990.

[25] A. Rizk and L. Sauter. Multicard: An open hypermedia system. In Proc. of the 4th ACM Conf. on Hypertext and Hypermedia (ECHT'92), December 1992.

[26] P.D. Stotts and R. Furuta. Petri-net-based hypertext: Document structure with browsing semantics. ACM Trans. on Information Systems, 7(1):3-29, January 1989.

[27] Berners-Lee T, R. Cailliau, J.F. Groff, and B. Pollermann. World-Wide Web: The information universe. Electronic Networking: Research, Applications and Policy, 1(2), 1992.

[28] F.W. Tompa. A data model for flexible hypertext database systems. ACM Trans. on Information Systems, 7(1):85-100, July 1989.

[29] J.D. Ullman. Principles of Database and Knowledge-base Systems, volume 1. Computer Science Press, 1988.

[30] P.T. Zellweger. Scripted documents: A hypermedia path mechanism. In Proc. of Hypertext '89, November 1989.