

# A Relational Model for Hypertext Structures

Dietrich Albert, Cord Hockemeyer, Theo Held

**Abstract**—Hypertext and hypermedia systems play an increasing role in the presentation of information. The probably most important of the rare formal approaches to describe the structure of a hypertext is the Dexter Hypertext Reference Model [1], [2]. Based on this model, we develop a mathematical model describing hypertext structures using mathematical relations.

This relational approach enables us to combine the model for hypertext structures with other mathematically formalized models. One example for such combinations is the relational algebra as the mathematical model underlying relational database theory and systems. Thus, we can use the efficiency of relational database systems and their selection routines in order to build adaptive hypertext systems.

**Keywords**— Relational hypertext model, Relational database system, Dexter model

## I. INTRODUCTION

THE first concept of a hypertext<sup>1</sup> system was developed by Bush [4] who proposed the *Memex* device for storing and linking information. The term *hypertext* itself was coined by Nelson. He defined hypertext as *a combination of natural language text with the computer's capacity for interactive branching or dynamic display [...] of a nonlinear text* [5]. Hypertext systems became broadly available with Apple's HyperCard system in the eighties (citemanual:hypercard and, more recently, with the World Wide Web [3].

In 1990, the National Institute of Standards and Technology (NIST) hosted a workshop on hypertext standardization where several reference models of hypertext were presented and compared [6]. Out of these models, the Dexter Hypertext Reference Model ([1], [2]; for an introduction see also Section II-A below) is the "most popular" one [7], [3].

Although the Dexter model is described formally using the Z-notation [8] this formalization is hardly usable for combination with other formally specified models. Therefore, we introduce an algebraic formalization of a relational hypertext model which is semantically almost the same as the Dexter model. This mathematical model — which is to our knowledge the first one describing the structure of hypertexts — enables us to combine the hypertext models with other models and theories. An important combination is that with relational algebra, the theoretical basis of

Dietrich Albert is Head of the Abteilung für Allgemeine Psychologie, Institut für Psychologie, Karl-Franzens-Universität Graz, Universitätsplatz 2/III, A-8010 Graz, Austria/Europe, E-Mail: Dietrich.Albert@kfunigraz.ac.at

Cord Hockemeyer works with Dietrich Albert, E-Mail: Cord.Hockemeyer@kfunigraz.ac.at

Theo Held worked with Dietrich Albert and is now at the Institut für Psychologie, Martin-Luther-Universität Halle, Brandbergweg 23c, D-06120 Halle, Germany, E-Mail: T.Held@psych.uni-halle.de

<sup>1</sup>We use the term *hypertext* also for other media like images or sound. One reason for doing this is the definition for hypermedia given by Rada [3].

relational database systems. This combination allows the efficient construction of adaptive hypermedia.

A second important combination which is rather directed towards the application of (adaptive) hypermedia systems is the Knowledge Space Theory from mathematical psychology [9], [10]. This combination (together with the relational algebra) leads to the construction of shells for intelligent tutoring systems as described by Albert and Hockemeyer [11].

In the following Section II, we first introduce the Dexter model and, afterwards, suggest a relational formalism for the structure of hypertext using the terminology introduced with the Dexter model. In Section III we describe how this relational model can be combined with relational database theory to efficiently determine sub structures of a hypertext. In Section IV, we compare our model with the Dexter model itself and with two widely used systems: HyperCard and HTML.

## II. A RELATIONAL MODEL OF HYPERTEXT STRUCTURES

The following formalization of hypertext and hypertext structures uses the terminology of the Dexter model. Our model does, however, not include the complete functionality required by Halasz and Schwartz [1], [2]. Thus, the complexity of the model is reduced. These restrictions of functionality are described in detail in Section IV-A where we also present concepts for formalizing the complete Dexter model. Since we focus on the description of the structure, all aspects dealing with the contents of hypertext documents are faded out. This results in a rather vague definition of those terms describing aspects of document contents, i.e. lying in the *within-component-layer* of the Dexter model.

### A. Overview of the Dexter model

In their Dexter model, Halasz and Schwartz [1], [2] distinguish three layers of a hypertext system: the *run-time layer*, the *storage layer*, and the *within-component layer*. The storage layer models the database describing the basic node/link structure of the hypertext. The nodes in a hypertext are called *components* in the Dexter model. The within-component layer models structures within the single nodes. The run-time layer models the presentation of the hypertext using the concept of *instantiations* of components and distinguishing the instantiations from the components themselves.

The fundamental concept in the storage layer is the component. There exist three kinds of components: atomic components, composite components, and links. An *atomic component* is what is often called a *node* or a *document* in a hypertext system. Such an atomic component consists of two parts: the *component info* and the *content*. The

latter is also called *base component*. The component info contains some *attributes*, a *presentation specification*, and a list of *anchors*. Anchors are end points of links which are introduced below. An anchor consists of an *anchor id* and an *anchor value*. The former is a label under which the anchor can be accessed, the latter describes the position of the anchor within the content of the component.

A *composite component* is very similar to an atomic component. The only difference is that the content of a composite component contains again other components. This concept is, e. g. realized by the `frames` recently introduced into HTML.

A *link* is a special kind of component which connects two or more components. It consists of a set of *specifiers* each of which contains a *component specification*, an *anchor id*, a *direction flag*, and a *presentation specification*. The component specification and the anchor id together specify an end point of the link. The direction flag determines whether the specifier describes a source or a destination of the link or both. The presentation specification of the link together with the presentation specification of the referred component are used by the presentation layer to determine the way the component will be presented if it is selected. This definition of a link allows a link to connect more than two components and it also allows bidirectional links.

### B. Elements of a hypertext

An atomic item in the structure of hypertext is the *base component*. A base component is a unit of information containing e. g. text, graphics, or sound. A *component* consists of a base component and some additional information, e. g. presentation specification or *anchors*. Anchors are the elements of hypertext structures which mark connections between units of information. We distinguish between *source anchors* and *destination anchors*. The links marked by the anchors are directed, i. e. they point from one *source anchor* to a *destination anchor*. When a component is presented to a reader, often only the source anchors are visible. They can be embedded in text or graphical material, or they can be presented separately, e. g. as buttons or menus. This may be specified in the presentation specification. Figure 1 shows an example of a base component and a corresponding component.

The component in the right part of this figure demonstrates three ways of presenting source anchors: the anchors `Information` and `Topic` are embedded in the text, the anchors `Next`, `Back`, and `Home` are presented as separate buttons, and the graphic diagram contains at least one anchor.

Destination anchors, as already said, are not visible in Figure 1. An example for a destination anchor is the name anchor tag in HTML.

*Definition 1:* Let  $B = \{b_1, \dots, b_n\}$  be a set of *base components*, let  $S = \{s_1, \dots, s_o\}$  be a set of *source anchors*, and let  $D = \{d_1, \dots, d_p\}$  be a set of *destination anchors*. The source anchors  $s \in S$  and destination anchors  $d \in D$  can be located on the base components.

A *component*  $c$  is a triple  $c = (b, S_c, D_c)$  constituted

by a base component  $b \in B$ , a subset  $S_c \subseteq S$  of source anchors, and a subset  $D_c \subseteq D$  of destination anchors. The set  $C = \{c_1, \dots, c_m\}$  can, therefore, be characterized as a subset of the Cartesian product of the set  $B$  of base components and the power sets  $2^S$  and  $2^D$  of source anchors and destination anchors, respectively:

$$C \subseteq B \times 2^S \times 2^D.$$

Note that due to the set properties of  $S_c$  and  $D_c$  an anchor can be located only once within one base component. However, it is, of course, possible to locate an anchor on several base components. Based on definition 1 above, we can now introduce the concept of a *link*. Links describe the connections between components.

*Definition 2:* Let the sets  $B$ ,  $S$ ,  $D$ , and  $C$  be defined as in definition 1 above. The set  $L = \{l_1, \dots, l_k\}$  of *links* is defined as a subset of the Cartesian product  $(C \times S) \times (C \times D)$  where each link  $l = ((c, s), (c', d)) \in L$  fulfills the condition

$$s \in S_c \text{ and } d \in D_{c'}.$$

A link  $l = ((c, s), (c', d))$  connects a source anchor  $s_c$  located on a component  $c = (b, S_c, D_c)$  with a destination anchor  $d_{c'}$  on a component  $c' = (b', S_{c'}, D_{c'})$ . A link can also connect two anchors located on one component, i. e.  $c$  and  $c'$  do not have to be different.

Based on the concepts defined above we now can give a formal definition of an *hypertext*.

*Definition 3:* Let  $B$  be a set of base components, let  $S$  and  $D$  be sets of source anchors and destination anchors, respectively, let  $C$  be a set of components, and let  $L$  be a set of links as specified in definitions 1 and 2 above.

The pair  $H = (C, L)$  is called a *hypertext*.

### C. Different types of links

There are two ways to characterize links: One is a characterization by the type of relationship between the linked components. We will define these types of links using properties of source anchors. The other way is a characterization by the type of the destination of the link. For this kind of characterization we will use different properties of destination anchors.

*Definition 4:* Let  $B$  be a set of base components, and let  $S$  and  $D$  be sets of source anchors and destination anchors, respectively. Furthermore, let  $C \subseteq B \times 2^S \times 2^D$  be a set of components, and let  $L \subseteq (C \times S) \times (C \times D)$  be a set of links according to definition 2.

Let  $p$  be a property which is applicable to source anchors. Then we denote by  $S_p$  the set of all source anchors which have this property  $p$ . A link  $l = ((c, s), (c', d)) \in L$  is called a *link of the source type  $p$* , if and only if the source anchor  $s$  has the property  $p$ , i. e. if  $s \in S_p$ . We write the set of all links of the source type  $p$  as  $L_p$ :

$$L_p = \{l = ((c, s), (c', d)) \in L \mid s \in S_p\}.$$

Correspondingly, if  $p'$  is a property which is applicable to destination anchors then  $D_{p'}$  is the set of all destination anchors with the property  $p'$ . A link  $l = ((c, s), (c', d)) \in L$  is called a *link of the destination type  $p'$* , if and only if the

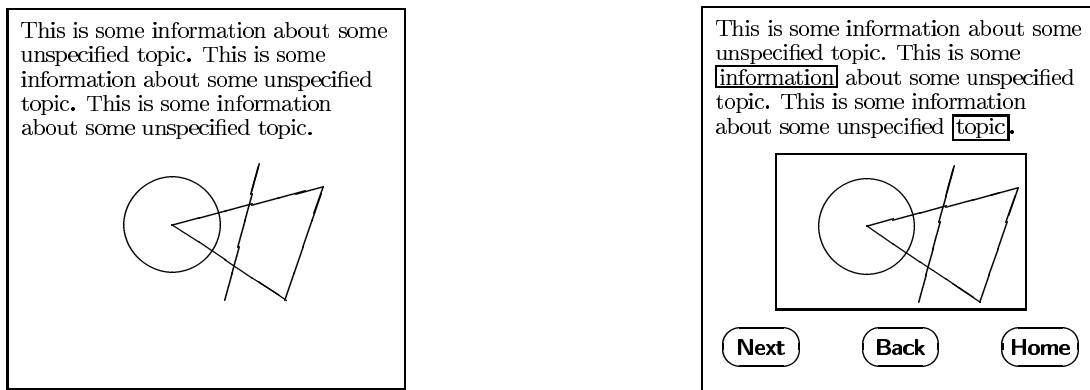


Fig. 1. Example of a base component and a corresponding component

destination anchor  $d$  has the property  $p'$ , i. e. if  $d \in D_{p'}$ . We write the set of all links of the destination type  $p'$  as  $L_{p'}$ :

$$L_{p'} = \{l = ((c, s), (c', d)) \in L \mid d \in D_{p'}\}.$$

Properties of source anchors can, for example, be “is an example of” or “is a glossary entry for”. Links with such source anchors are called *example links* or *glossary links*. Examples for properties of destination anchors are “is a program” or “is a video”. Links with such destination anchors are called *program links* or *video links*, respectively.

These different types of links may be used to determine if source anchors and other parts of the component should be visualized to the user. This decision would depend on the type of the link via which the component was reached. In an intelligent tutoring system, e. g., a component might contain a problem together with some hints for its solution and the solution itself. If this component was selected via an *example link* problem and solution could be shown. If it was selected via a *training link* the problem and an input mask for its solution could be presented, together with a **(Help)** and a **(Solution)** button. However, if it was selected via a *test link* only the problem and the input mask would be visible to the pupil.

Besides that, e. g. Rada [3] suggests that links of different types should be presented in different ways. A user of an intelligent tutoring system could then use this knowledge about the type of the link when he/she decides whether or not to follow the link. The number of different styles must, of course, be restricted so that the user is not overburdened.

#### D. Formal properties of link sets

Links in a hypertext describe relationships between the single components. These relationships can be described as relations on the set of components. In definition 5, we define a relation between linked components.

**Definition 5:** Let  $H = (C, L)$  be a hypertext as defined in definition 3. We define the binary relation  $\vdash \subseteq C \times C$  on the set  $C$  of components. For any components  $c, c' \in C$ , we obtain  $c \vdash c'$  if and only if there exists a link  $l = ((c, s), (c', d)) \in L$ .

In the following we transfer several basic properties of binary relations from the relation  $\vdash$  from definition 5 to the

set of links by reducing these properties to appropriate properties of sets of links. The general definition of these properties of relations can be found e. g. in [12].

**Proposition 1:** Let  $H = (C, L)$  be a hypertext as defined in definition 3 and let  $\vdash$  be the relation given in definition 5.

(i) The relation  $\vdash$  is *reflexive* if each component  $c \in C$  has a source anchor  $s \in S_c$  and a destination anchor  $g \in D_c$  such that  $((c, s), (c, d)) \in L$ .

(ii) The relation  $\vdash$  is *antisymmetric* if, for any components  $c, c' \in C$ , if there exist two links  $((c, s), (c', d))$  and  $((c', s'), (c, d'))$  in  $L$  for some source anchors  $s, s' \in S$  and some destination anchors  $d, d' \in D$ , the condition  $c = c'$  holds.

(iii) The relation  $\vdash$  is *connected* if, for any components  $c, c' \in C$  there exists a link  $((c, s), (c', d))$  or a link  $((c', s), (c, d))$  in  $L$  for some source anchor  $s \in S$  and some destination anchor  $d \in D$ .

(iv) The relation  $\vdash$  is *transitive* if, for any two links  $((c, s), (c', d))$  and  $((c', s'), (c'', d')) \in L$ , there also exists a link  $((c, s''), (c'', d'')) \in L$ .

Using the properties of relations, we can also define conditions for  $\vdash$  to be an order.

**Definition 6:** Let  $H = (C, L)$  be a hypertext as defined in definition 3 and let  $\vdash$  be the relation given in definition 5.

(i)  $\vdash$  is called a *quasi-ordinal* set of links if  $\vdash$  is reflexive and transitive.

(ii)  $\vdash$  is called a *partial ordinal* set of links if  $\vdash$  is reflexive, antisymmetric, and transitive.

(iii)  $\vdash$  is called a *linear ordinal* set of links if  $\vdash$  is connected, antisymmetric, and transitive.

(iv)  $\vdash$  is called an *antichain* if, for any  $c, c' \in C$ ,  $c \vdash c'$  if and only if  $c = c'$ .

Some of these properties of link sets may seem inappropriate. Linear order, e. g. is merely the opposite of the hypermedia idea. It may, however, be used to describe a subset of the hypermedia structure which describes the user’s history. Regarding hypertext-based intelligent tutoring systems, however, there are several applications of these properties.

We may, for example, consider *prerequisite links* pointing from a component to its prerequisites, i. e. to other components which teach knowledge that is needed to understand the actual component’s content. Such informa-

tion are useful to determine which components should be made available to a pupil. These prerequisite links should form a transitive set of links since the prerequisites of prerequisites of a component are obviously also prerequisites of this component itself. However, the author of such a hypertext-based intelligent tutoring system will probably specify only some of these prerequisite links. In the system actually used later, this set of prerequisite links should then be closed under transitivity. Of course, there are also other uses for the properties of link sets defined above, e.g. the identification of parallel exercises or instructions.

### E. Navigation paths

Moving from one component to another following the hypertext links is called *navigation*. The *navigation path* is the history of a user, i.e. the sequence of components visited.

*Definition 7:* Let  $H = (C, L)$  be a hypertext. An ordered  $n$ -tuple  $P = (c_1, c_2, \dots, c_n) \in C^n$  is called a *path* if and only if for all  $i = 1, \dots, n-1$  there exists a Link  $l \in L$  such that

$$l = ((c_i, s), (c_{i+1}, d))$$

for some  $s \in S_{c_i}$  and  $d \in D_{c_{i+1}}$ .

We denote by  $\mathcal{P}_n \subseteq C^n$  the set of all paths  $P \in C^n$  which can be chosen while navigating through a hypermedia structure in  $n$  steps.

There are two important application for paths. One of them is the user's history, i.e. the sequence of components the user has visited so far. The other application is the preselection of a path through the hypermedia structure by its author. Both paths enable the creation of *forward*- and *back*-links. In many systems, the forward- and back-links through the user's history are offered by the hypertext system through special buttons which are not part of the component presented. The forward- and back-links specified by the author, on the other side, are in general part of the specific component. In this way, the two different types of forward- and back-links can easily be distinguished.

Considering hypertext-based intelligent tutoring systems, the navigation paths of the pupils get additional importance: while in general the navigation paths are used to help the hypertext user in navigating through the hypertext, they can in intelligent tutoring system also serve to estimate the pupil's knowledge and, in consequence, to decide which components and links may be presented to the pupil depending what s/he has learned so far.

## III. FLEXIBLY SELECTING SUB HYPERTEXTS USING RELATIONAL DATABASE THEORY

In this section we show how the definitions from Section II-B and II-C can be used to define a relational database. For this application, only few, minor changes must be applied. First, we give a short introduction to relational algebra and relational database theory in general. Afterwards, we show how a database describing the structure of a hypertext can be defined using the definitions from Section II.

### A. Basics of relational algebra and relational database theory

The following introduction to basic concepts of the relational database model and relational algebra follows largely the terminology and notation used by Elmasri and Navathe [13]. The example relations used in this section are simplified versions of those suggested in Section III-B.1.

In the relational database model, data in a database is represented as a set of relations. Each of these relations can be viewed as a table where a tuple of the relation can be interpreted as a row in the table. A *relation schema* REL is a set  $\{A_1, \dots, A_n\}$  of attributes. Each *attribute*  $A_i$  has a *domain*  $D_i = \text{dom}(A_i)$ . These domains may not have a structure to be evaluated by the database system. A relation schema describes a relation whose *name* is  $R$ . The number  $n$  of attributes in a relation schema is called *degree of the relation*.

In the terminology of relational databases, a relation schema defines the structure of a table, and the attributes denote the names of the table's columns. For each attribute, a domain is specified, i.e. a set of possible values for this attribute. A relation  $R$  following a schema REL is a subset of the Cartesian product of the domains of the attributes.

*Example 1:* The relation schema

DOC (DID, BASE, PRES)

describes a relation (or table) DOC with three attributes (or columns):

- DID Unique identifier of the document (e.g. a number),
- BASE Base component building the content of the document (cf. Fig. 1,
- PRES Presentation specifications (e.g. colour, size, or background for the presentation).

The domains of the attributes are the set of document identifiers, the set of base components, and the set of all subsets of presentation specifications, respectively. This relation scheme describes a first approach to a *document relation scheme*. This new *document* concept is a component without anchors but it differs from the base component by the additional presentation information. As such, it can be located between base component and component in the Dexter model. The components themselves cannot be represented explicitly in relation schemas since this would require to include the sets of source and destination anchors. These sets of anchors, however, would introduce a structure which is later to be used by the database system which is not allowed in relational database theory as we have said above. We elaborate the concept of a document in more detail in Section III-B.1.

In the example above, the attribute DID has the special property of uniqueness, i.e. for a relation  $D$  according to the relation schema DOC and for any two tuples  $t = (i, b, p), t' = (i', b', p') \in D$ , we obtain  $t = t'$  if and only if  $i = i'$ . The DID can be realized e.g. by a unique document number. An attribute satisfying this condition is

called a *key*. A key can also consist of a subset of attributes, e.g. of two attributes whose combination is unique within a relation. Being a key is a property of a relation schema which imposes a corresponding constraint onto each relation following the schema.

Based on the definitions given above the *relational algebra* provides operators for manipulating and combining relations. There exist two groups of operators. One of these groups contains general set operators: *union*, *intersection*, *difference*, and *Cartesian product*. The other group consists of operators which have been developed especially for relational databases: *select*, *project*, and *join*. The general set theoretical operations *union*, *intersection*, and *difference* have the same meaning in the relational algebra and, therefore, do not need any further explanation. The *Cartesian product*, however, is slightly different from the general set theoretical operator: If  $R \subseteq M_1 \times M_2 \times \dots \times M_m$  and  $S \subseteq N_1 \times N_2 \times \dots \times N_n$  are two relations then their *Cartesian product*  $T = R \times S$  in relational algebra can be described by  $T \subseteq M_1 \times \dots \times M_m \times N_1 \times \dots \times N_n$ , i.e. we get a set of  $(m+n)$ -tuples instead of pairs of  $m$ - and  $n$ -tuples. The tuples  $t \in T$  are exactly defined by the equation

$$T = \{(m_1, \dots, m_m, n_1, \dots, n_n) \mid (m_1, \dots, m_m) \in R \text{ and } (n_1, \dots, n_n) \in S\}.$$

In the following, we illustrate the operators which have been developed especially for relational algebra.

The *select* operation

The *select* operation selects a subset of tuples from a relation which satisfies a given condition. We write a selection operation in the following form:

$$\sigma_{\langle \text{selection condition} \rangle}(R)$$

where  $R$  is a relation and  $\langle \text{selection condition} \rangle$  is a Boolean expression on comparisons of different attributes or of attributes and constants. The resulting relation has the same relation schema as the original relation  $R$ .

*Example 2:* Consider the relation schema DOC from Example 1 and let  $D$  be a relation with schema DOC. The expression  $\sigma_{\text{DID}=x}(D)$  returns a relation with the relation schema DOC which contains all tuples from  $D$  which have a value of  $x$  in their DID component. Since DID is a key attribute in DOC, the resulting relation contains at most one tuple.

The relation returned by  $\sigma_{\text{BASE}=b}(C)$ , on the other hand, contains all documents which have the base component  $b$ .

The *project* operation

While the *select* operator selects specific rows of a table, the *project* operator selects specific columns. For a relation  $R$  with a relation schema REL, the term

$$\pi_{\langle \text{attribute list} \rangle}(R)$$

describes a relation with a relation schema as described in  $\langle \text{attribute list} \rangle$ . This  $\langle \text{attribute list} \rangle$  describes a

subset of the set  $\{A_1, \dots, A_n\}$  from the relation schema REL. The order of the attributes in the resulting relation schema is determined from the order of the attributes in the  $\langle \text{attribute list} \rangle$  and it is independent from the order in the original schema REL.

*Example 3:* Reconsider the relation schema DOC from Example 1 and let again  $D$  be a relation according to relation schema DOC. The expression  $\pi_{\text{BASE}}(D)$  returns a relation whose relation schema contains only the attribute BASE, i.e. you get a table containing just this one column. Note, however, that the resulting relation of a projection may also contain less rows than the original relation since there may be a loss of uniqueness due to the reduction of attributes. In the above example, a base component may describe the content of several different documents. The resulting relation of the projection would contain this base component only once since it is a relation and, as such, a set.

The *join* operation

The operators described so far are used to reduce relations. The *join* operator is used to combine them. The resulting schema contains all attributes from both original schemas. A join differs, however, from the Cartesian product by the restriction that the resulting relation contains only those tuples which satisfy a given condition. For two relations  $R$  and  $S$  with relation schemas REL and SEC, we write a join as

$$R \bowtie_{\langle \text{join condition} \rangle} S$$

where  $\langle \text{join condition} \rangle$  is a Boolean expression of comparisons of the form  $A_R \Theta A_S$ ,  $A_R$  and  $A_S$  are attributes of REL and SEC, respectively, which have the same domain, and  $\Theta$  is a comparing operator in this domain.

*Example 4:* Let DOC be the relation schema from Example 1 and let  $D$  be a relation with schema DOC. Let

$$\text{LINK}(\text{LID}, \text{SD}, \text{SA}, \text{DD}, \text{DA})$$

be a relation schema with the following attributes:

- LID Unique identifier of the link,
- SD Identifier of the source document,
- SA Identifier of the source anchor,
- DD Identifier of the destination document,
- DA Identifier of the destination anchor,.

and let  $L$  be a relation of schema LINK. The expression

$$L \bowtie_{\text{DD}=\text{DID}} D$$

returns a relation  $R$  with the schema

$$\text{RS}(\text{LID}, \text{SD}, \text{SA}, \text{DD}, \text{DA}, \text{DID}, \text{BASE}, \text{PRES}).$$

All tuples in the relation  $R$  have identical values in the DD and DID document. The resulting relation  $R$  contains for each link also the information about the destination document. However, links whose destination document does not exist are eliminated.

Although the conditions in the *join* operator are not limited to equality comparison this is a very frequent case. The

resulting relation contains two equal columns which means a redundancy of information. Therefore a new operator, the *natural join* was created. The natural join

$$R *_{(\langle \text{list}_1 \rangle), (\langle \text{list}_2 \rangle)} S$$

returns a new relation whose schema contains all attributes from  $R$  and those attributes from  $S$  which are not contained in  $\langle \text{list}_2 \rangle$ . The new relation contains all combinations of tuples  $r \in R$  and  $s \in S$  where  $r$  has the same values in the attributes of  $\langle \text{list}_1 \rangle$  as  $s$  in the corresponding attributes of  $\langle \text{list}_2 \rangle$ .

*Example 5:* Consider once more the relation schemas DOC and LINK from the Examples 1 and 4 together with the relations  $D$  and  $L$ . The expression

$$L *_{(\text{DD}), (\text{DID})} D$$

returns a relation  $R$  with a relation schema

$$\text{RS}(\text{LID}, \text{SD}, \text{SA}, \text{DD}, \text{DA}, \text{BASE}, \text{PRES}).$$

The tuples in this relation  $R$  contain for each link  $l \in L$  additionally the base component and the presentation specification of the destination document. The expression

$$R = L *_{(\text{LID}, \text{SD}, \text{SA}, \text{DA}), (\text{LID}, \text{DD}, \text{SA}, \text{DA})} L$$

determines a relation  $R$  which contains all links  $l \in L$  which connect two anchors within one document. The schema of the resulting relation does, however, not contain the DD attribute. The comparison in this natural joint is done on a per document base. The link identifiers (LID) are combined to ensure that we combine links which themselves. The SD and DD attributes are combined to select the links between anchors within one document. The SA and DA attributes are each compared to ensure that they occur only once in the resulting relation scheme.

In this case, there would also be another, more efficient way to compute the relation  $R$ :

$$R = \pi_{\text{LID}, \text{SD}, \text{SA}, \text{DA}}(\sigma_{\text{SD}=\text{DD}}(L)).$$

Actually, this equation describes a rather straightforward way: first we select all those links where source and destination document are equal. In a second step, we then eliminate the DD attribute by a projection.

### B. Using a relational database for constructing sub hypertexts

In this section, we first suggest a relational database design describing the structure of a hypertext. Afterwards, we give some examples how this database could be used to select sub structures according to special needs.

The database design suggested differs in one point from the formalization given in Section II: Since the attributes may not have a structure to be evaluated within the relational algebra, the *component* cannot be represented explicitly. Therefore we introduce the concept of a *document* which contains a base component plus additional information but no anchors.

### B.1 Design of a relational database describing hypertext structures

The following relation schemas are very close to those specified in Example 1 and 4: The central schema is the document. As already said before, this schema does not contain information about anchors which are stored in a separate relation. On the other side, there are two new attributes: the *document identifier* DID and the *presentation specification* DPRES. The document identifier DID is a key attribute of DOC.

The new relation schema ANCHOR has five attributes: the *anchor identifier* AID, the *document identifier* DID, the *position descriptor* POS, the *direction* DIR, and the *property descriptor* PROP. Currently, the form of position and property descriptions is kept open. The attributes AID and DID together build a key in the ANCHOR relation schema. This means that there are no two different tuples in a ANCHOR relation which share a common AID and a common DID. However, there may be different anchors which share the same anchor identifier but which have different document identifiers. An anchor identifier “TOP”, for example, might exist in several documents. Please note that equally labeled attributes in different relations (like DID in DOC and ANCHOR) actually denote the same fact — in this case the document identifiers in the ANCHOR relation refer to those used in the DOC relation. The DIR attribute refers to the distinction between *source* and *destination* anchors.

The third schema is the link which differs from the Example 4 only by the additional attribute LPRES. This attribute contains another presentation specification. The actual presentation of a document is determined by combination of the document’s presentation specification and the specification of the link by which the document was reached. In the LINK relations, the LID attribute is a key attribute. To sum up, we get the following three relation schemas:

$$\begin{aligned} &\text{DOC}(\text{DID}, \text{BASE}, \text{DPRES}), \\ &\text{ANCHOR}(\text{AID}, \text{DID}, \text{POS}, \text{DIR}, \text{PROP}), \text{ and} \\ &\text{LINK}(\text{LID}, \text{SD}, \text{SA}, \text{DD}, \text{DA}, \text{LPRES}). \end{aligned}$$

This database design realizes the formalization described in Section II with the exception of the separated ANCHOR relation. Although we use the terminology of the Dexter model [1], [2], the model and the database design presented so far do not satisfy the conditions of the Dexter model. We dropped some properties because we do not need them and we receive a simpler model. In Section III-B.2 and IV-A, we describe possibilities to extend the database design to full dexter functionality.

### B.2 Examples for applying relational databases for selecting hypertext sub structures

One important property of hypertexts is the maintenance of consistency by disabling the creation of *dangling links*, i.e. links with a non existing end point. This is a property which Halasz and Schwartz [1], [2] require for hypertexts but which we do not want to realize. With this decision,

we follow the argumentation of Grønbaek and Trigg [14] (cf. Section IV-A). Assuming three relations  $D$ ,  $A$ , and  $L$  according to the schemas DOC, ANCHOR, and LINK, respectively, from the section above we can eliminate all dangling links in the following way: In a first step we determine an ANCHOR relation  $A'$  which contains only anchors with actually existing documents:

$$A' = \pi_{AID, DID, POS, DIR, PROP} (A *_{(DID), (DID)} D)$$

The *link* operation determines all anchors whose document exists and the following *projection* reduces the schema of the interim relation to the ANCHOR schema. In a second step, we compute a LINK relation containing only those links where source and destination anchors exist in the specified source and destination documents, respectively:

$$L' = (L *_{(SD, SA), (DID, AID)} (\pi_{DID, AID}(A'))) *_{(DD, DA), (DID, AID)} (\pi_{DID, AID}(A'))$$

In this second step, we join the LINK relation twice with a reduced ANCHOR relation  $A'$ . The ANCHOR relation  $A'$  is reduced by a projection to the DID and AID attributes, i. e. by omitting the POS, DIR, and PROP attributes. Since only those attributes are left which are used for the join condition, the LINK relation scheme keeps untouched by the *join* operations. Finally, we keep in the inner *join* only those tuples (i. e. links) which have an existing source anchor and, from these links, we keep in the outer *join* only those which have an existing destination anchor. Since the attributes AID and DID together build a key in the ANCHOR relation scheme, there is at most one tuple in  $L'$  for each tuple in  $L^2$ .

Another task may be the elimination of isolated documents, i. e. documents which are neither source nor destination of a link. If we have three relations  $D$ ,  $A$ , and  $L$  according to the relation schemas DOC, ANCHOR, and LINK, respectively, we can determine the non-isolated documents by the following operation:

$$D' = (D *_{(DID), (SD)} (\pi_{SD}(L))) \cup (D *_{(DID), (DD)} (\pi_{DD}(L)))$$

The first part of the above expression determines all documents which are source of at least one link, and the second part computes all documents which are destination of at least one link. With this new DOC relation  $D'$ , one can now eliminate dangling links in order to get a consistent and connected hypertext. Analogously to the elimination of isolated documents, it is possible to eliminate isolated anchors. In this case, the (AID, DID) pairs of the ANCHOR relation have to be joined with the (SA, SD) and (DA, DD) pairs of the LINK relation.

An application oriented usage of the relational databases can be found in the field of tutoring hypertexts. Assume we have a tutorial hypertext with lessons, examples, test

<sup>2</sup>In this operation, it would, of course, also be important to make sure that source and destination of the links actually are source and destination anchors, respectively. We skip this step here since it would make the operation too complicated.

and training problems, and a glossary. Exercises in this system might be written in a way that the same document can be used as an example, as a training problem, or as a test problem. This usage would be determined by the property (PROP) of the source anchor through which the document is reached. A user might now be interested to have a smaller version which contains just the lessons and the glossary but not the exercises. To achieve this, we can just select a subset of anchors

$$A' = \sigma_{(PROP=lesson) \vee (PROP=glossary)}(A).$$

Afterwards, we can remove dangling links and, finally, isolated documents and anchors in order to receive a smaller system adapted to this user's needs.

#### IV. RELATIONSHIP TO EXISTING SYSTEMS AND MODELS

In this section, we look for the Dexter model and for some existing hypertext systems (HyperCard and HTML) up to what extent we can model these systems or how the model could be extended to meet them. In this comparison, we focus on the mathematical model presented in Section II but we also refer to the database approach from Section III-B.

##### A. The Dexter hypertext reference model

There are four major properties for a hypertext system which Halasz and Schwartz [1], [2] demand in their Dexter model and which are not satisfied by our model. The first property is the prevention of dangling links. Following the argumentation of Grønbaek and Trigg [14], we decided to allow dangling links. Besides that, for the database approach, we have shown in Section III-B.2 that dangling links can be eliminated easily.

A second property that our model lacks, so far, is the usage of multi-anchored links. We did not yet incorporate this feature in our model since it makes things more complicated. In the mathematical model, multi-anchored links could be realized as sets of pairs containing a component and an anchor. This results in a definition of the set  $L$  of links as

$$L \subseteq 2^{C \times A}.$$

The set  $A$  of anchors used in this equation is the union of the sets  $S$  and  $D$  of source and destination anchors from Section II. Additionally, there may also be bidirectional anchors according to the Dexter model.

Regarding the database approach, it is quite simple to define a new relation schema

$$\text{DEXTER} - \text{LINK} (\text{LID}, \text{AID}, \text{DIR}, \text{LPRES})$$

with the attributes

LID	Link identifier,
DID	Document identifier,
AID	Anchor identifier,
LPRES	Presentation specification.

In this schema, the LID attribute is not a key, i. e. there are several tuples in such a DEXTER – LINK relation which have

the same LID. Actually, all tuples in this relation sharing a common LID describe together one link.

A third point is that, opposed to the Dexter model, we do not regard links as a kind of components in our mathematical model. This is of interest regarding composite components, which is the fourth point. Here, Halasz and Schwartz [1] are a little vague: Originally, they distinguished between atomic, link, and composite base components, and they defined a composite base component as a sequence of base components. Later ([2]), they moved this definition up to the component level, i.e. they defined a composite component as a component whose base component contains other components. Both definitions would allow a link to be part of a composite component. If we do not regard links as some kind of component, the former definition of composite (base) components is no problem in our model, since we make no assumptions about the inner structure of base components. In the latter case, i.e. defining composite components as components whose base components contain complete components, the problem of anchor identifier uniqueness comes up. For this problem, there are two possible solutions: we can require a global uniqueness of anchor identifiers. or we can re-map anchor identifiers within composite components. Another problem occurs in the database approach where base components containing components would contradict the prohibition of structured attributes in relational algebra.

## B. HTML and HyperCard

In this section, we want to go briefly into two hypertext systems which have found quite a broad distribution: HTML and HyperCard. HyperCard was for some time distributed together with Apple Macintosh Computers and, as a consequence, is used a lot. HyperCard was one of the hypertext systems which influenced the Dexter model [1], [2]. A newer system is the *World Wide Web* (WWW) with its *Hypertext Markup Language* (HTML), a worldwide hypertext system with components distributed across the Internet.

### B.1 HTML

A major difference are the distinction between components and links and between components and base components in the model. In HTML, there is always one HTML document which contains the content, presentation specifications, anchors, and links. We think, however, that this is only a difference in the representation and that, internally, the HTML documents just contain the information completely. Of course, this means in the case of database operations as they have been described in Section III-B.2, the information on anchors and links would have to be extracted from the HTML documents before and to be incorporated again after performing the database operations. However, this should be no problem, since those operations will probably be applied off-line and, therefore, are not time critical.

A second interesting point is the frame concept in HTML. This concept is close to the concept of composite components in the Dexter model discussed in Section IV-A above.

The third point we want to mention here, is the possibility to produce HTML pages online via the *Common Gateway Interface* (CGI). This can be viewed as a temporal change of the hypertext.

Like our relational model, HTML also allows the creation of dangling links. As already mentioned before, we prefer the freedom to specify links to not yet existing anchors to the security of an always consistent hypertext.

### B.2 HyperCard

The HyperCard system [15] made hypertext available to a broader public. While HTML gives much freedom to the presenting browser, a hypertext in the HyperCard system consists of a set of *cards* (i.e. components) which have an exactly specified appearance. In principle, both, the Dexter model and our relational model can be applied to HyperCard. There are, however some functionalities HyperCard does not offer. Cards in HyperCard are grouped in *stacks*, all cards belonging to one stacks share some features like the size. This concept contradicts to the idea of composite components.

Building cards directly with HyperCard the creation of dangling links is impossible. This does, however, not hold for the creation of cards using HyperCard's script language HyperTalk [16].

## V. DISCUSSION

In this paper, we have presented a formalism for the description of hypertext structure. This formalism can be transformed directly into a relational database definition. We have given some examples of how such a relational database describing the structure of a hypertext can be used to determine sub hypertexts in the field of hypermedia based intelligent tutoring systems. Our next goal is the implementation of these ideas in a evaluable system.

An important point for further research is the user model and the adaptation of the system to the user. In Section II-D, we used the user's history as an example for the user model. Regarding hypermedia-based intelligent tutoring systems we suggest to combine our relational model of hypertext structure with the *knowledge space theory* (see, e.g., [9], [10], [17]).

### Acknowledgments

This work was supported by the grant ERBCH-BICT941599 in the program on Human Capital and Mobility (HCM) of the Commission of the European Communities.

## REFERENCES

- [1] Frank Halasz and Mayer Schwartz, "The Dexter hypertext reference model," In Moline et al. [6], pp. 95-133.
- [2] Frank Halasz and Mayer Schwartz, "The Dexter hypertext reference model," *Communications of the ACM*, vol. 37, no. 2, pp. 30-39, 1994.
- [3] Roy Rada, "Hypertext, multimedia and hypermedia," *The New Review of Hypermedia and Multimedia*, vol. 1, pp. 1-21, 1995.
- [4] Vannevar Bush, "As we may think," *The Atlantic Monthly*, July 1945, Also online available: <http://www.isg.sfu.ca/~duchier/misc/vbush/>.

- [5] Theodor Holm Nelson, "Getting it out of our system," in *Information Retrieval: A Critical Review*, George Schechter, Ed., pp. 191–210. Thompson Books, Washington D.C., 1967.
- [6] Judi Moline, Dan Benigni, and Jean Baronas, Eds., *Proceedings of the Hypertext Standardization Workshop*, vol. 500–178 of *NIST Special Publications*, Gaithersburg, MD 20899, January 1990. National Institute of Standards and Technology.
- [7] Kaj Grønbaek and H. Trigg, Randall, "Hypermedia system design applying the Dexter Model," *Communications of the ACM*, vol. 37, no. 2, pp. 26–29, Feb. 1994, Guest editorial for a special section on hypertext.
- [8] J. M. Spivey, *The Z Notation*, Prentice Hall International, 1987.
- [9] Jean-Paul Doignon and Jean-Claude Falmagne, "Spaces for the assessment of knowledge," *International Journal of Man-Machine Studies*, vol. 23, pp. 175–196, 1985.
- [10] Jean-Claude Falmagne, Mathieu Koppen, Mike Villano, Jean-Paul Doignon, and Leila Johannesen, "Introduction to knowledge spaces: how to build, test and search them," *Psychological Review*, vol. 97, pp. 201–224, 1990.
- [11] Dietrich Albert and Cord Hockemeyer, "Dynamic and adaptive hypertext tutoring systems based on knowledge space theory," in *Artificial Intelligence in Education: Knowledge and Media in Learning Systems*, Benedict du Boulay and Riichiro Mizoguchi, Eds., Amsterdam, 1997, vol. 39 of *Frontiers in Artificial Intelligence and Applications*, pp. 553–555, IOS Press.
- [12] Peter A. Fejer and Dan A. Simovici, *Mathematical Foundations of Computer Science*, vol. 1, Springer Verlag, 1990.
- [13] Ramez Elmasri and Shamkat B. Navathe, *Fundamentals of Database Systems*, Addison-Wesley World Student Series. The Benjamin/Cummings Publ., Redwood City, CA, 1989.
- [14] Kaj Grønbaek and H. Trigg, Randall, "Design issues for a Dexter-based hypermedia system," *Communications of the ACM*, vol. 37, no. 2, pp. 41–49, Feb. 1994.
- [15] *HyperCard Reference*, Santa Clara, CA, 1990.
- [16] *HyperCard Script Language Guide*, Santa Clara, CA, 1990.
- [17] Dietrich Albert and Theo Held, "Establishing knowledge spaces by systematical problem construction," in *Knowledge Structures*, Dietrich Albert, Ed., pp. 78–112. Springer Verlag, New York, 1994.