Retrieval of Volatile Database Output Through Hypermedia Applications

Geert-Jan Houben and Paul De Bra
Eindhoven University of Technology, Department of Computing Science
PO Box 513, 5600 MB Eindhoven, the Netherlands
{houben,debra}@win.tue.nl

Abstract

This paper reports on research on the generation of hypermedia (Web) applications for querying multimedia databases. In [5] we have proposed a heuristic algorithm for generating navigational structures for multimedia database output, based on ideas from RMM [6].

Here we discuss two extensions to our approach. The first extension concerns in more depth the generation of navigation for volatile database output. The semantics of the data are used to produce a navigational structure for the hypermedia presentation, while by adding elements to the query syntax we allow the user to influence this. The second extension deals with the automatic generation of a presentation for the multimedia database output. While many of the presentation issues can be settled at design time by the designer of the application, a number of issues should be tackled at query time. These issues concern the presentation of records and their relationships with other records.

1. Introduction

In many application areas relational databases are used in the implementation of information systems. Traditionally the data stored in those databases have an administrative, textual nature. The standard access mechanisms that are available to the users are based on the elegant and simple data structures underlying the relational model. Most of those access mechanisms are implemented using relational query languages such as SQL. Due to the often strongly structured and purely textual nature of the data this approach is sufficient to support the users in their manipulation of data from the database.

We consider applications such as employee databases, museum databases, geographic information systems, and mail-order catalogs and services. In this class of applications one typically encounters data that are less structured and not purely textual. The data contain for example pictures, where relationships exist between data elements such as a piece of text that builds a caption for a certain picture. In the purely relational databases designers would try to remove such relationships from their design during the process of normalization. In modern database applications the data are not as strongly structured as is usually the case in traditional relational databases.

In dealing with legacy applications we consider information systems that continue to use relational databases as the prime means of storage for the information. The prime function of such an information system is to retrieve data from the database as the result of a query and present this information to the user in a manner that is easily comprehensible. Due to the nature of the data the traditional query mechanisms do not suffice anymore for this specific class of applications. The use of a *hypermedia platform* such as World Wide Web can help to represent the less structured and not purely textual information. For our target class of applications the use of a hypermedia platform can help to build an easily accessible front-end for the data retrieved from relational databases.

In developing hypermedia applications for data from relational databases one can distinguish between two cases. On the one hand there is information for which we are able to design an application beforehand. This means that during the design phase of the system, long before queries are asked, a designer is able to thoroughly think about the way in which the information is captured in a hypermedia format. On the other hand there is information resulting from queries that not have been foreseen, and for which no design had been possible. A consequence is that the hypermedia format must be generated at the time of querying. Our research described in this paper concentrates on this second type of information: data retrieved as the result of an arbitrary query for which no design has been explicitly prepared. Due to the nature of these data, we speak of "*volatile*" data or "volatile" database output. In generating a

0-7695-0001-3/99 $10.00 (c) 1999 IEEE

hypermedia format for the presentation of these data we want to obtain a format that increases the ease and comprehension for the user, while it acknowledges that the volatility of the data prevents an intensive design phase.

In [5] we have described an approach to generating hypermedia applications for volatile database output. To produce a complete hypermedia application design our method includes directives for the (semi-)automatic generation of both navigational structures ("relationships between data elements") and data presentation ("contents of data elements"). The design and construction of a hypermedia application involves the representation of relationships between information objects: one important aspect of a hypermedia application is the *navigational structure* that can be used to inspect the information objects by "navigating" from object to object. Our hypermedia application development approach is therefore based on the ideas of RMM [6], which combines elements from the Entity-Relationship model [3] and HDM [4] to effectively manage relationships between objects. Certain ideas from RMM are, however, not applicable in our context. We must extend the RMM approach due to the dynamic and volatile nature of the data that are used. While RMM is meant to facilitate the manual design process of a hypermedia application, our goal is to automatically construct a hypermedia application for volatile database output. The information contained in a query result generally depends on the data structures underlying the volatile information. RMM, and its supporting tool RMCase [2], do not help in the case of answers to arbitrary queries. The main problem is that the dynamically determined structure of a query result cannot be (trivially) translated into a hypermedia format.

Besides the generation of a navigational structure the development method must support the generation of the *data presentation.* A hypermedia application does not only consist of (hyper)links between data elements, the data elements themselves should be presented in an elegant way. For "existing" data elements, those data elements of which we explicitly know beforehand that they can be asked for such as records from base relations, the design of the presentation appears feasible beforehand, e.g. specified in the definition of those data elements. For "volatile" data elements such as derived records a definition of the presentation beforehand is not feasible, so the generation of the presentation must be included in our generation method.

Our generation method specifies how the hypermedia representation, consisting of the navigational structures and the data presentation, is generated. The core of our approach is to (re)use as much as possible the representation aspects that are embedded in the data definition. Moreover, we look at the query specification to see if the user has included explicit directives on how to generate the hypermedia presentation. Note that for the end user specification of explicit presentation directives we use an extension of SQL. However, that does not imply that the end users must actually use that extended SQL: they will have the possibility to specify the queries (including presentation directives) through an elegant (Query-By-Example-like) user interface.

We use certain *heuristics* for the generation of the hypermedia application on the basis of data definition and query specification. These heuristics are the result of a combination of engineering decisions and results from first investigations into the user satisfaction with alternative hypermedia designs.

In this paper we recall and extend the description of our hypermedia application development method:

- We show how more of the heuristics can be used to generate satisfactory application designs. Using several examples we illustrate how query specifications can be translated into hypermedia designs.
- For the generation of navigational structures we look at additional extensions to the query language. Using a preprocessing of the query specification we are able to generate navigational structures for the query result guided explicitly by the user.
- We also discuss aspects that are related to the presentation of the information (the records in the query result). This includes both the presentation of the "contents" of the records, and the (hyper)links and other navigational mechanisms between records.

2. Navigation representation

2.1. Navigational structures

Navigational structures build the most characteristic aspect of a hypermedia application. In a hypermedia application the links between the nodes (containing the data elements) represent the structural relationships between the data elements. The motivation for using the hypermedia representation is to increase the ease with which users can access the information (originally stored in relational databases). In the applications that we consider we are dealing with volatile data that are the result of queries. In order to represent these volatile data in an easily accessible manner our application generation method specifies how navigational structures are produced. We use heuristics that guide the design process to produce navigational structures for the volatile data to be represented.

We borrow aspects of RMM [6] in order to express the relationships between the data elements. With RMM we are able to specify the mechanisms available for connecting the data elements and thus offering user

access to the data elements.

The navigational structures that we typically encounter in the data retrieved from the relational databases are a combination of hypermedia access routines that relate records to each other, *inter-record navigation*, and access routines that relate parts of a single record to each other, *intra-record navigation.*

2.2. Inter-record navigation

Based on RMM the mechanisms and tools available for inter-record navigation are:

- index
- guided tour
- indexed guided tour

An *index* is used to access records by referring to some key identifier. By choosing an identifier the user asks the system to navigate to the corresponding record. Typically the index is composed of words or icons that naturally identify the associated record.

A *guided tour* is used to access records in a given order. The records are connected in a chain like manner, and the user can follow that chain and thus access the different records.

An *indexed guided tour* is a combination of an index and a guided tour. Essentially the user has both options to choose from: navigating using the identifiers in the index, or following the predefined path of the guided tour.

In comparison to RMM our approach does not include other inter-record navigation mechanisms. This is mainly motivated by the fact that the volatile data collections that result from asking queries are not really different with respect to the inter-record connections: the data collections are always *sets* of records.

The only difference that seems appropriate is that the user will be able to specify how records are displayed on "pages" in the hypermedia environment. In a hypermedia application we can display one record on one (hyper)page and offer (hyper)links to navigate between such pages. In practice it is often convenient to display a number of records on one page, if the size of the representation of the records allows this. We could use this functionality to offer an additional record connecting structure: however, we do not address this functionality in this paper.

2.3. Intra-record navigation

Just as in RMM we use *slices* as a way to divide the presentation of a record in multiple parts that each can be presented in a natural and easily comprehensible manner. Since the presentation of all properties (attributes) of one record on one hypermedia node (page) often appears not to be feasible because of size restrictions, the designer can divide the record presentation in separate parts.

These slices offer a view on a part of a record. In order to move the view towards other parts of the record, there is a link structure available that connects the slices of a record. These *slice links* build the intra-record link structure, that is orthogonal to the inter-record connections used to collect sets of records. The definition of this slice link structure is determined at the level of relations: all records of a relation share the same intra-record navigation structure.

Besides specifying the slices of a record and the links between those slices, we have to determine how the inter-record navigation and intra-record navigation are connected. RMM defines that the slice structure has a *head slice*, which means that one slice is specified that becomes visible at the time the user "enters" the presentation of the record: the navigation inside the record starts from that head slice.

The next figure shows how in our prototype system the two kinds of navigation are realized on the screen of the user. We see an index and a guided tour (represented by the bus icon) to represent the inter-record navigation between professor records, while within the selected record we see a slice with information on the professor and some course information, and we also see links to other slices to represent the intra-record navigation.

2.4. Navigational structure specification

To automatically generate navigational structures we need a definition for the design of the structures. In this section we address the issue of the specification of that design. The general heuristics underlying our approach specify that the navigational structures of the query result depend on the structure embodied in the data (definition) and on the query specification itself.

The most important principle used in our heuristics is

that we use the predefined design as much as possible. If we want to represent the *data* contained in the result of a query, we start the generation of the design of the navigational structures by considering the definition of these data (in the data dictionary). Only when the user explicitly specifies that this data definition should be *overridden*, then the initial design is modified in accordance with the specific commands issued by the user at the time of querying. Note that this is completely in the spirit of RMM: we want to use as much as possible the outcome of the effort from the human designer.

There can be multiple reasons why a user wants to deviate from the standard design. For example, the user knows that the query result contains only a limited set of records, such that a different representation is more suited than the general design from the data dictionary. It can also be the case that the user purposely wants to use the query result to present the information in a different way. In general we assume that deviations occur when the user is an advanced user with enough insight into the application and its underlying database to be able to adjust the representation of the query result.

The overriding of the general design rules by the users is possible by allowing the users to include specific *design commands* in the query specification. Thus, the specification of queries in our approach serves two purposes. Not only are the data determined that are to be represented, but at the same time the navigational structures are determined that are needed to "connect" the data to each other. In the sequel we will show how the query syntax also facilitates this second purpose.

While eventually the specification of a query will take place in a friendly user interface, for the moment we use SQL as the language in which the advanced user is able to specify queries. It does not necessarily mean that end users must specify their queries and design commands in this extended SQL. They will have the possibility to use a user-friendly Query-By-Example-like user interface that enables them to specify the necessary details in a convenient manner.

Note that we make a distinction between the representation of existing data and new volatile data. For data that are simply retrieved from the base tables of the database the design generation method tries to copy as much as possible the initial design for the records from those base tables, unless explicitly overridden. For new volatile data structures the method must produce a design with navigational structures that incorporate natural (inter-record and intra-record) navigation mechanisms.

Example 2.1

As an example we consider the following situation. Suppose that a query asks for a subset of records from a base table *employees*:

select name, age, photo
from employees
where city = "Eindhoven";

Assume also that the designer of the database has defined that the records of *employees* are displayed using an indexed guided tour. The heuristics initially direct us to display the records contained in the result of this query using an indexed guided tour. It means that an indexed guided tour is created that allows the user to navigate between the records from the query result.

If the user wants to override the initial choice for an indexed guided tour, considering that an index may seem more appropriate, then the user must add the necessary commands to the query specification:

select name, age, photo
from employees **index**
where city = "Eindhoven";

3. Single slice queries

Definition 3.1

A single slice query is a query in which the **from** *clause contains only one relation and in which the* **select** *clause only contains attributes from a single slice of that relation.*

In order to produce a representation of the result of a query two aspects need to be considered. First, the representation of the set of selected records has to be determined. Also, the representation of the individual records has to be chosen.

Using the principle that we want to reuse the original data definition as much as possible, the heuristics decide that the representation for (the navigation within) a set of records from a single base table is copied from the definition for that table. This means that for standard selection and projection queries we can copy the inter-record navigation design from the definition.

According to our heuristics only the specific slice is important or relevant. Unless the user specifies otherwise, the system will choose a representation for the retrieved records based on the representation of that specific slice.

Definition 3.2

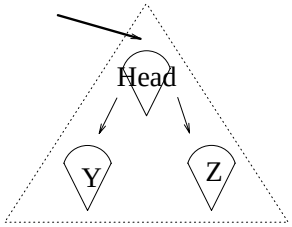
The result of a single slice query is composed using the following rules:

- *the system chooses as the inter-record navigational structure for the query result the same structure as is explicitly defined by the designer for the given base table, unless explicitly overridden by the user*
- *the system chooses the (complete) slice of which the attributes occur in the* **select** *clause as the head slice for the query result, unless explicitly overridden by*

the user

Example 3.1

Consider a relation R with attributes ABCDEFGHIJK. The attributes of R are organized in a slice structure as follows: the head slice contains attributes ABCD, there exists a slice Y containing EFG and a slice Z containing HIJK; both Y and Z are reachable through a (forward) slice link from the head slice.



Assume that the user specifies the following query:

select A,B,C,D
from R
where C = 0;

Our heuristics first detect that the user is interested in a selection of records of R, so for the inter-record navigation the system chooses the defined representation for R: if the data definition states that the table R is accessed through an index, then this set of R records is also accessed through an index.

Secondly, the heuristics detect that the user asks for information from the head slice of R, so the method decides to produce a record representation (and intra-record navigation) based on the presentation of this head slice: whenever the user "enters" one of the retrieved records, the user is presented with the head slice of that record.

Note that when we speak about the presentation of a slice, we mean that also the intra-record links accessible from that slice are actually available for the user. In this example it means that the ABCD slice is shown, together with links to slices Y and Z.

Example 3.2

Consider the same relation R and the following query:

select E,G
from R;

Now the heuristics direct the system to again use the standard definition for R for the inter-record navigation. For the intra-record navigation the observation is made that the attributes in the **select** clause are taken from slice Y. The heuristics specify that the user will be presented with Y as the new head slice. Although not all attributes from Y are mentioned in the **select** clause, according to the heuristics the user asks for that complete slice.

These examples show that for the intra-record navigation the application offers existing slices, where it depends on the **select** clause which slice acts as the head slice. Note that the presentation includes the given slice links, so if a slice is chosen without any given outgoing links (like slice Y in the last example), then no other slice is accessible.

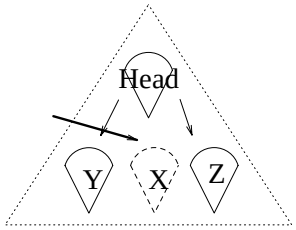
The examples also show that (for this class of single slice queries) "existing" objects, slices and records from base tables, are produced. Even when the **select** clause does not include all of the attributes of a slice, then the method produces that complete slice. The user can override that choice by explicitly asking for a "new, volatile slice", a slice that is not predefined but exists only in this volatile query result.

Example 3.3

Consider the next query, similar to the previous one:

select (E,G) **asslice** X **newhead**
from R;

Using the **asslice** command, the user specifies that a new, volatile slice is to be made. If necessary, the user can give a name for that new slice, in this case X. The consequence is that the existing intra-record structure for the records of R is augmented with an additional slice X, while this slice X becomes the new head (entry) slice.

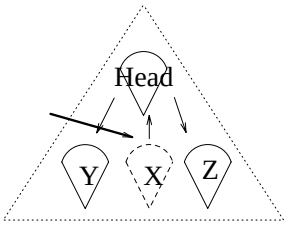


Example 3.4

In the previous example the user is not able to navigate from X to another slice of R. With the **links** command it is possible to ask the system to explicitly create additional links.

select (E,G) **asslice** X **newhead,**
links (X,**head**(R))
from R;

This query does the same as the previous one, only this query offers an additional link between slice X and the head slice of an R record.



We see that we can introduce "new" slices on the basis of old slices. It is possible that multiple new slices are created. We are also able to define new volatile relationships within the data: both for connecting the new data to the old data, as to specify other relationships than the ones foreseen. In order to specify a new navigational structure we can specify additional links and, possibly, a new head (entry) slice

Example 3.5

Consider the next query:

select (H,K) **asslice** X, (I,J,K) **asslice** W,
links (**head**(R),X), (X,W)
from R;

This query produces two volatile slices X and W (out of a single original one) and specifies the access from the original head slice of R.

4. Multiple slice queries

Definition 4.1

A multiple slice query is a query in which the **from** *clause contains only one relation and in which the* **select** *clause contains attributes from multiple slices of that relation.*

Basically, the heuristics are the same when multiple slices are involved in the **select** clause. The prime difference is that it appears difficult to choose one of the slices as the new head slice. Therefore, the basic choice is to keep the original head slice.

Definition 4.2

The result of a multiple slice query is composed using the following rules:

- *the system chooses as the inter-record navigational structure for the query result the same structure as is explicitly defined by the designer for the given base table, unless explicitly overridden by the user*
- *the system chooses the original head slice as the head slice for the query result, unless explicitly overridden by the user*

Example 4.1

Consider the following query with multiple slices appearing in the **select** clause:

select A,B,I,J,K
from R;

In this query attributes are specified from two slices. Our heuristics lead to a intra-record navigational structure in which the user gains access to those slices from the standard head slice: this means that we present a record by showing its head slice and (included with that slice) the links from that head slice to the other slices.

Example 4.2

The next specification is a similar one:

select (A,B) **asslice** X, (I,J,K) **asslice** W,
links (**head**(R),X), (X,W)
from R;

In this specification the user explicitly asks for two new, volatile slices X and W with exactly those attributes. Moreover the user specifies how the new slices are attached to the given link structure. The head slice remains the original one.

Example 4.3

Consider the following version:

select (A,B,I,J,K) **asslice** X **newhead**
from R;

Here, attributes from different slices have been packed together into one new volatile slice. This slice is also declared to be the new head slice. Using **links** (X,**head**(R)) the user could have added a link from the new volatile slice X to the given slice link structure.

Whether we are dealing with one or more slices, these queries have in common that we are dealing with records from a single table. This implies that we can copy practically all design aspects from the data definition. The representation of the new, volatile data can be easily derived from the representation of the original data, as the inter-record navigation can stay the same.

5. Multiple relation queries

In this section we consider queries that involve records from multiple tables. Specially, *join* operations use records from several tables to build temporary records composed out of the given records. The most prominent difference between *multiple relation queries* and *single relation queries* is that in the representation of the result of a multiple relation query inter-record and intra-record navigation are integrated.

Definition 5.1

A multiple relation query is a query in which the **from** *clause contains multiple relations.*

In general, when we have a query with multiple relations in the **from** clause, the user specifies combinations (for example, triples when 3 relations occur behind **from**) of records. These records can possibly be associated with each other through the **where** clause. Even when there is no **where** clause, we obtain combinations of records. So, in order to represent these combinations the system must derive a representation for those "volatile relationships" between records. For our

heuristics the only important matter is the construction of the "combined" record that gives access to all the mentioned slices.

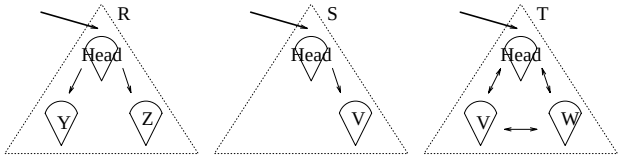
Definition 5.2

The result of a multiple relation query is composed using the following rules:

- *the system chooses as the inter-record navigational structure for the query result the same structure as is explicitly defined by the designer for the first relation mentioned in the* **from** *clause, unless explicitly overridden by the user*
- *the system chooses as the navigational structure within a combined record an indexed guided tour that connects the slice structures of the composing records, unless explicitly overridden by the user*
- *the system chooses as the intra-record navigational structures the original structures of each of the composing records, unless explicitly overridden by the user*

Example 5.1

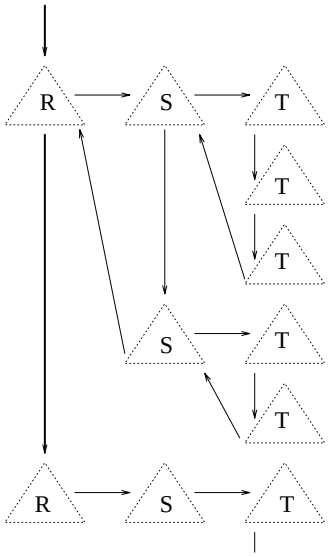
Consider the relation R from the previous examples (we denote an attribute Z from R as R.Z) and the relations S and T. Suppose that S has two slices: a head slice U with attributes ABCD and a slice V with attributes EFG. Assume there is a link from S.U to S.V. Suppose T has three slices: a head slice U with attributes ABC, a slice V with attributes DE and a slice W with attributes FGHI. Assume that between all slices of T there are bi-directional links.



The following query symbolizes a typical query involving multiple relations:

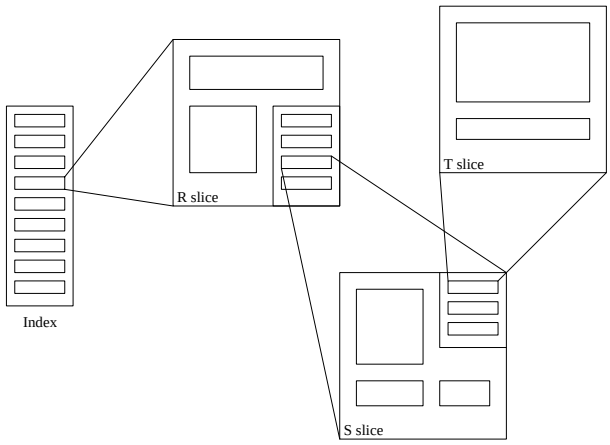
select R.A, R.B, R.C, R.D, S.E, S.F, S.G, T.D, T.E
from R,S,T
where R.A=S.A **and** S.A=T.A;

This query combines records from three tables. Our method produces a representation for the combined records. The set of selected records is accessible through the standard inter-record access mechanism defined for R (the first relation). The connection between "joined" records is established using indexed guided tours to "glue" S and T records to R records (based on the selection condition). The indexed guided tour is chosen, as it is the most versatile of the navigation mechanisms.



Assume that an R record r1 is combined with two S records s1 and s2, and that r1 and s1 are combined with three T records t1, t2 and t3. When we access the query result (through an index), we can "enter" record r1. From record r1 we can go into its own slice structure, starting at its head slice, but we can also follow an indexed guided tour that leads us to first s1 and then s2. When we enter record s1, we can decide to inspect its own slice structure, but we can also follow an indexed guided tour that connects t1, t2 and t3. Thus we can inspect all combinations that involve r1.

The next figure shows the elements that are visible on the user's screen: an index for the joined records, the R slice of the joined record, its S slice and its T slice.



Example 5.2

The next example shows a query where new volatile slices are created per composing record:

select (R.A, R.B) **asslice** R.X **newhead**,
(S.D, S.E, S.F) **asslice** S.Y, T.D, T.E,

links (R.X,**head**(R)), (**head**(S),S.Y)
from R,S,T
where R.A=S.A **and** S.A=T.A;

Now, the relevant information from the R record is packed together in one new slice R.X, while the relevant information from the S record is packed together in a new slice S.Y. Note that new slice links are added and that the new slice S.X is appointed as the new head slice. Note also that the information from the T record is left as it is.

So, if the user wants to inspect only parts of the composing records, it is possible to create new slices specifically for these parts.

Example 5.3

The next query creates a single slice with all the information from the composing records united:

select (R.A, R.B, S.D, S.E, S.F, T.D, T.E) **asslice** Z,
links (**head**(R),Z)
from R,S,T
where R.A=S.A **and** S.A=T.A;

So, a new volatile slice is produced that is reachable from R's head slice. Our heuristics say that such a volatile slice is added to the slice structure of the record from the first relation, here R. It means that, in comparison to the previous example, we again have the same combined slice structures (combined through an indexed guided tour), but this time the first slice structure contains an additional slice with the mentioned attributes. There is an important consequence: since the new volatile slice unites information from the three combined records, there are multiple instances of that slice associated to that single R record. If we consider an R record r that is associated with totally 7 pairs of an S record and a T record, then we obtain 7 combinations involving r. So, we obtain 7 new Z slice instances. As a consequence the original inter-record navigation is changed: instead of the record r occurring once in the index used to access the R records, r's place is taken by 7 instances of the record each with another volatile slice Z.

In general, when we create new volatile data that involve records from multiple relations we can choose between accessing these data through slices associated with base records or through slices that combine attributes from different base records.

Example 5.4

Consider the following more complex query:

select (R.A, R.B) **asslice** R.X **newhead**,
(S.D, S.E, S.F) **asslice** S.Y,
(R.A, R.B, S.D, S.E, S.F, T.D, T.E) **asslice** Z,
links (**head**(R),Z), (Z,**head**(R)),
(**head**(R),R.X), (**head**(S),S.Y)
from R, S **index**, T **index**
where R.A=S.A **and** S.A=T.A;

The user creates separate slices R.X and S.Y, while a new united slice Z is created (and attached to the R record as well). One other interesting feature is the use of the **index** command after the second and third relation name, S and T. This overrides the standard use of an indexed guided tour to connect S and T records to an R record. In this case with every R record an index is associated that gives access to all corresponding S records, and similarly for the connection to all corresponding T records.

The motivation for creating volatile slices is that the user expects that the isolation of the information into a new slice makes the data more comprehensible. Our heuristics primarily stimulate the user to access the records through their original intra-record navigation structures. The heuristics mainly produce inter-record navigation mechanisms to access the "right" combinations of records. Their purpose is not to create totally new objects. Our target queries are used to retrieve existing base records and straightforward combinations thereof.

6. Slice presentation

6.1. Slice design

In this section we turn to the representation of the information contained within records and more specifically within slices of records. We could say that in the previous sections we have addressed the navigation between slices, while now we turn to the layout of slices and navigational mechanisms. Note that in this paper we do not actually consider the graphical aspects of the layout: we focus on how the generation of the layout can help to improve the presentation of volatile data.

Whenever we want to generate a presentation for record slices, we must consider two aspects:

- the presentation of the *data* contained in the slice
- the presentation of the *navigational* mechanisms, which can be divided into
  - intra-record links
  - inter-record links

In the next section we will discuss the presentation of the navigational mechanisms. This section considers the presentation of the slices themselves.

We already know that the level of slices is the lowest level at which presentation of information can be designed using RMM. This grouping of attributes into slices is based on knowledge of the semantics of the attributes. It is used to divide a record into information chunks that can be elegantly displayed by a hypermedia (World Wide Web) browser, without too much

information becoming “invisible” due to a lack of space on the screen. Typically, slice design is an activity that takes place beforehand, when the data properties are defined: not at the moment of querying.

In our approach we assume that a designer of a hypermedia application has specified in the phase of definition how all slices explicitly contained in the data structures are to be presented. First, the designer chooses a presentation for every attribute contained in the slice. Then, a slice presentation is constructed using the chosen attribute presentations. For our purposes the construction of the attribute presentations are outside our scope. The construction of slice presentations on the basis of attribute presentations is within our scope: for the predefined slices the designer specifies this construction.

When we are dealing with a *volatile* slice, we want to use the available slice presentation definitions in the design of a presentation for this unforeseen slice. Our approach should construct slice presentations for volatile slices that result from a query. This means that we cannot be satisfied with predefined and hard coded slice presentations in a language like HTML. The most characteristic property of volatile slice presentations is that we need to construct a slice presentation based on the attribute presentations for those attributes that are contained in the slice. This is a consequence of the principle of “maximal reuse”. However, there is no trivial way how the original attribute presentations are packed together into a slice presentation. The most important design issue is how the different attribute presentations are displayed relative to each other: a very simple (too simple!) solution would be to display the different attributes from left to right (or top to bottom) in the order of the **select** clause. So, we need a reusable specification of the relative positions of attributes, in order to encode this properly in HTML.

Example 6.1

Suppose we ask a query in which we create a new volatile slice containing three attributes: a photo, a subtitle and a description (piece of text). We want to be able to design the slice in such a way that the subtitle attribute is positioned under the photo attribute (and that it is centered under the photo), and that the description attribute is positioned to the right of the photo.

Our approach recognizes that such a slice design is not something that is specified at the time of querying. However, we want to be able to specify a number of conditions beforehand, that are used at the time of querying to produce an acceptable design. This implies that a translator script combines size information and positioning constraints into a layout that is then converted to HTML code. Work on using constraints for automatic user interface layout generation has already been performed by Borning et al. [1].

6.2. Identification attributes

Another issue involved in the presentation of records and slices is the choice and presentation of *identification attributes* used to provide navigation to records and slices. For example, an index is a list of identifiers that each give access to a record (or, more specifically, its head slice). In an index for person records we can think of the use of the names of the persons or the use of small photos. Similarly, the slice structure inside a record is implemented in such a way that from each slice a number of other slices are accessible. For example, for person records a slice presentation for the head slice can contain a number of icons, such as a photo icon to go to the photo and video slice, a book icon to go the favorite books slice, a bike to go the hobby slice, etc. An identifier attribute is not necessarily an attribute itself. Often it is a compact icon that generally symbolizes a key attribute of the slice.

The design of head slices and identification attributes is extremely important as they play a significant part in the process of retrieving information. In general, identification attributes and head slices are used to identify the records the user is searching for. Thus, the combination of identification attribute and head slice builds a *compact* representation of the record, that is used in the first stage of the retrieval process.

In the second stage of the retrieval process, the *complete* representation (with all other slices) is used to supply the user with all the information available on the selected records. Then, the user can inspect the selected records more in detail.

Using these two types of representations our approach facilitates a *two phase information retrieval process*. In the first phase record presentations based on the compact slice presentations are used to identify the records that are (potentially) interesting. In the second phase the complete presentations for those records are used to make a thorough inspection of the information possible.

7. Navigation presentation

Until now we have concentrated on the definition of which navigational structures exist between records and between slices. Now, it is time to discuss the design of the presentation of the navigation mechanisms. For the complete presentation of the query result to the user it is essential that the presentation of the navigation mechanisms and the presentation of the slices themselves are generated simultaneously, because both parts of the presentation share (fight for) the same screen real estate.

Our heuristics incorporate three basic principles:

- Different “frames” are used for navigation and information.
- The designer fixes the maximum space used for the navigation (and information) frames.
- The designer defines how the different navigation mechanisms are displayed exactly.

When a query result is displayed, the user must navigate through a set of records. Our approach directs the system to use a frame that displays the inter-record navigation mechanism, for example an index, and it uses a frame that displays the “current” record. This record is the record that the user has “selected” through the use of the navigation mechanisms: at the start the current record is simply the first record from the query result.

In the case of combined records, for example with a join, we can obtain a pair of frames for each composing record. If we are dealing with a join of three relations, we get three *information frames* and three *navigation frames*. The three information frames each show one of the composing records. The first of the navigation frames offers the inter-record navigation through the set of combined records, while the second and third navigation frame present the index connections between the associated records from the different tables. Note that the second and third navigation frame disappear when the combination is realized through a guided tour only: we put the guided tour icon on the information frame itself.

Above we have said that an information frame shows a record. In fact it shows the current slice of the record: at the start that is the head slice. In the information frame the user can choose to follow slice links, expressed by identifier attributes, towards other slices of that record. We allow the designer to specify the position of these slice links in a separate place in that frame, for example in a subframe. Note that the design of the presentation of the identifier attribute is part of the slice presentation.

Besides deciding which frames exist, the designer is asked in our approach to specify the maximum space for the different frames. Specially for the navigation frames we want to make sure that they do not become too large and that they thus prevent the information frames from showing the real data the users want to see.

Also, the designer specifies how the different navigational structures are displayed. This means that the designer defines how an index, a guided tour, etc. are displayed, and they can do that for each relation differently.

Much of the slice and navigation presentation is dealt with at design time: this means that less work is needed in the generation of the presentation at query time. To a certain extent this holds just as much for volatile information as for existing information. For instance, if a query results in a single indexed guided tour through a set of new slices, the presentation of that result can (according to our heuristics) be the same as that for single indexed guided tours through a set of existing slices (of the same records).

8. Conclusion and further work

This paper discusses two directions of extension to the approach presented in [5]. The first direction deals with the generation of *navigation* for volatile database output. We have shown how the semantics implicitly included in the data is used to derive a navigational structure consisting of inter-record and intra-record links. We have also described how the users can influence that generation by including explicit directives in the query specification.

The second direction of extension is concerned with the *presentation* of slices and of relationships between slices. We have shown that a lot of design issues can be settled beforehand, at the time of defining the data. A number of issues remain to be tackled at the time of querying. We have described how for the generation of an elegant presentation of volatile slices, the designer has to specify the different presentation aspects.

We are currently involved in the implementation of the different heuristics and in experiments (user trials) to validate the approach. These experiments should give more insight in the quality of the heuristics for both the generation of the navigational structures and for the production of the presentations of slices and their relationships.

References

[1] A. Borning, K. Marriott, P. Stuckey, Y. Xiao, “Solving linear arithmetic constraints for user interface applications”, *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1997.

[2] A. Díaz, T. Isakowitz, V. Maiorana, G. Gilabert, “RMC: A Tool to Design WWW Applications”, *Proceedings of the Fourth International World Wide Web Conference*, Boston, 1995, pp. 559-566.

[3] R. Elmasri, S. Navate, *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, 1990.

[4] F. Garzotto, P. Paolini, D. Schwabe, “HDM - A Model for the Design of Hypertext applications”, *Proceedings of the ACM Hypertext'91 Conference*, 1991, pp. 313-328.

[5] G.J. Houben, P. De Bra, “World Wide Web Presentations for Volatile Hypermedia Database Output”, *Proceedings of WebNet97, the World Conference of the WWW, Internet, and Intranet*, AACE, Toronto, 1997, pp. 229-234.

[6] T. Isakowitz, E. Stohr, P. Balasubramanian, “RMM: A Methodology for Structured Hypermedia Design”, *Communications of the ACM*, Vol. 38, No. 8, 1995, pp. 34-44.