I Krajowa Konferencja Naukowa Technologie Przetwarzania Danych,
Poznań, 26-28.09.2005

Virtual Schemas in Visual Interfaces to Databases

Mariusz Trzaska*

Abstract. Virtual Schemas are part of the visual information retrieval system called Mavigator. The system is dedicated to naive users (computer non-professionals) and allows them to retrieve information from any data source, in particular, from object-oriented and XML-oriented databases. A significant problem related to all browsing tools is a way of presenting database schema graph. In many cases, there is a need to customize data-base schema, in particular, changing some names, adding new associations or removing some classes. Virtual Schemas allow creating a virtual customized version of existing database schema and navigate within the database according to this schema. Schema customization options include creating virtual association and virtual attributes, hiding/showing particular classes and renaming some schema items.

1. Introduction

Virtual schemas are a kind of database views implemented on application side rather than in database itself. That approach assures flexible modifications of database schema independently of information coming from the database.

Database views are subject of research and development for a very long time in the database community. However up to now, there are few proposals for object-oriented or XML-oriented environments, which are implemented, powerful and easy to use. This is mainly caused by hard problems, including updates of virtual objects delivered by views. Such a view mechanism has been implemented within visual database browsing prototype called Mavigator [15], [16], [17]. The corresponding module, called Virtual Schemas, represents read-only views in a visual browsing interface.

In Virtual Schemas views are properties of user applications. By giving the possibility to create views on the application side rather than on the database side it is possible to achieve the following qualities:

- Hide some parts of a schema, e.g., for security reasons,
- Connect via virtual associations classes that are not physically connected,

* Polish-Japanese Institute of IT, Koszykowa 86, Warsaw, Poland, e-mail: mtrzaska{at}pjwstk.edu.pl

- Define associations that select objects, i.e., by using a kind of a WHERE statement,
- Hide some intermediate classes – it is especially useful when one works with a relational data source where some many-to-many relationship is represented by two one-to-many tables and an intermediate table,
- Change some names, e.g. for some business reasons, for customization, or just for translating them into another language.

Such possibilities could be performed by ordinary database views, but in majority of cases we do not have access/rights to modify data sources or simply data sources have no views capabilities.

Virtual Schemas are based on one of the successful approaches to database views employing SBA (Stack-Based Approach) [5], [13], [14]. My proposal allows one to define a new schema, which consists of virtual associations, virtual attributes and classes (collections). All the items are expressed in the query language SBQL (Stack-Based Query Language). Classes reflect physically collections stored in the database and cannot be virtual. This is mainly caused by the fact that Mavigator al-lows to record objects, which are selected by the user. Due to this assumption we avoid situation when a user records a virtual object being a combination of information from several classes. Because it might happen that the user switches his/her search to a new defined ad hoc schema, some of the virtual classes may become invisible and their definition can be lost. However another scenario is permitted: the user can record an object and then can switch to a new schema where the class is hidden. In such a case a special message will be send to the user. On contrary, defining virtual associations and virtual attributes is not limited in any way. It is possible to use every SBQL construct valid in the appropriate context.

The remainder of this paper is organized as follows. In Section 2 we discuss related work. In Section 3 an overview of the Mavigator's information retrieval capabilities is given. Section 4 contains the detailed description of the proposal. Section 5 concerns implementation and architecture of the prototype. Section 6 concludes.

2. Related Work

To my best knowledge Mavigator is a first system, which fully implements database views on the visual browser side, rather than in a database management system server. Usually graphical tools for information retrieval have a rich set of retrieval capabilities and just work with a particular data source; no the user-side tuning of a tool through views is provided. On the other hand, systems dealing with views are just database servers and have no or very modest facilities for visual information retrieval.

One of the most advanced examples of views' uses in visual tool is AMOS [4] and its graphical browser GOOVI [2] developed by Cassel and Risch. This system uses mediators, which are counterparts to the views. In contrast to Mavigator, GOOVI browser is independent tool, which only collaborate with AMOS. In the other words, browser does not have views capabilities, but only works with the system, which supports them.

A frequent problem with different data sources is a common data model. The AMOS developers have decided to build an object-oriented extension of DAPLEX [8]. AMOS covers various aspects of views: joining heterogeneous data source, retrieving/updating data and ability to work with functions/methods. The first property has been achieved by implementing special wrappers in programming languages such as C, Java, and Lisp. Unfortunately GOOVI browser is quite simple i.e. selecting objects is done via textual query editor.

Another interesting browser is Watson [10], which is dedicated to Criminal Intelligence Analysis. It is based on an object graph and provides facilities to make various analyses, in particular, retrieving all objects connected directly/indirectly to specified objects (i.e. all people, who are connected to a suspected man), finding similar objects, and so on. Querying capabilities include filtering based on attributes and filter patterns. The latter allow filtering links in a valid path by their name, an associated type, a direction or a combination of these methods. The manner of work is similar to the metaphor that is called extensional navigation. However Watson's designers use term *views* not in the same context as we are. Comparing to my term, their *view* is some state of the extensional navigation, i.e. some number of objects, which are connected by various links. Watson's view has nothing in common with a database schema.

Taking into consideration just information retrieval capabilities, without using views, there are a lot of interesting systems. Some of them are based on graphical query languages, some on graphical browsing interfaces, and others have both functionalities. An example is Pesto [1] that has the possibilities to browse through objects from a database. Otherwise to Mavigator, the browsing is performed from one object to a next object. For instance, the user can display a Student class object, but to see another student, he/she needs to click next (or previous) button and replace current visualization. Besides browsing, Pesto supports quite powerful query capabilities. It utilizes the query-in-place feature, which enables the user to access nested objects, e.g. courses of particular students, but still in the one-by-one mode. Another advantage concerns complex queries with the use of existential and universal quantification. Such complex features may however compromise usability for some kinds of users and kinds of retrieval tasks.

Typical visual querying system are Kaleidoscape [6], based on its language Kaleidoquery [7], and VOODOO [3]. Both are declared to be visual counterparts of ODMG OQL thus graphical queries are first translated to their textual counterpart and then processed by an already implemented query engine. The first one uses an interesting approach to deal with AND/OR predicates. It is based on a flow model described previously by Schneiderman [9]. We find it very useful and intuitive thus we have adopted it to the metaphor.

Polaris [11], designed for relational databases, has some querying capabilities, but it seems that the major emphasis has been put on data visualization.

3. Information Retrieval Methods

To fully understand concepts and applications of the virtual schemas, it is necessary to get picture of the Mavigator's information retrieval methods.

Mavigator is made up of three metaphors utilized for information retrieval: intensional navigation, extensional navigation and persistent baskets. Next subsections contain detailed description of the methods.

3.1. Intensional navigation

Intensional navigation utilizes a particular virtual schema (Figure 1), which consists of the following primitives:

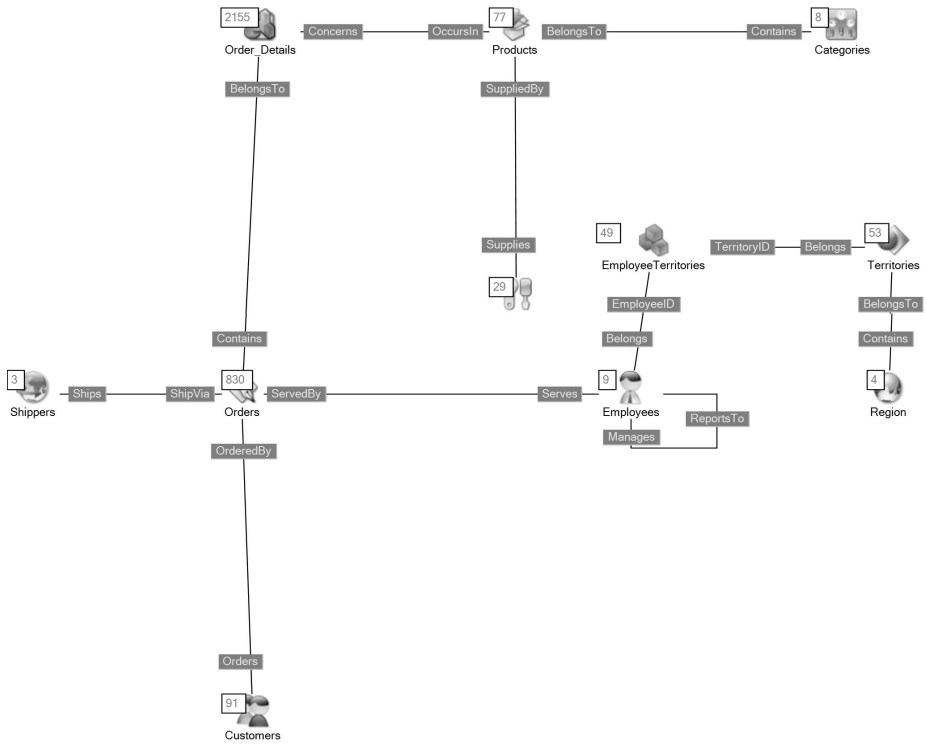

Figure 1. Example of a database virtual schema

- Vertices, which represent classes or collections of objects. With each of them we associate two numbers: the number of objects that are marked by the user (see further) and the number of all objects in the class,
- Edges, which represent semantic associations among objects (in UML terms),

- Labels with names of association roles. They are understood as pointers from objects to objects (like in the ODMG standard, C++ binding).

The user can navigate through vertices via edges. Objects, which are relevant for the user (candidates to be within the search result) can be marked, i.e. added to the group of marked objects. There are a number of actions, which cause objects to be marked:

- Filtering through a predicate based on objects' attributes. The action cause marking those objects for which the corresponding predicate is true. Filtering objects through a predicate is analogous to the SQL *select* clause.
- Manual selection. Using values of special attributes from objects (identifying objects by comprehensive phrases) it is possible to mark particular objects manually.
- Navigation (Figure 2) from marked objects of one class, through a selected virtual association role, to objects of another class. An object from a target class became marked if there is an association link to the object from a marked object in the source class. Figure 2 explains the idea. Let's assume that the *Firm* set of marked objects has four marked objects: *F1,…,F4*. Than, navigating from *Firm* via *employs* cause marking eight objects: *E1,…,E8*. This activity is similar to using path expressions in query languages. A new set of marked objects (the result of navigation) replaces existing one. It is also possible to perform a union or intersection of new marked objects with the old ones. Notice that the later options allow one to perform a query like: get all employees working in the "Main" department **and** earning more than 5000. The options are easy to understand (even for naive users) and greatly enhance the retrieval capabilities.

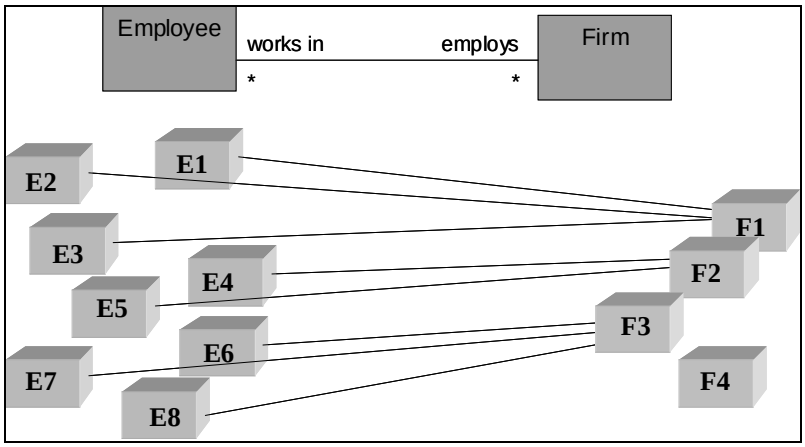

Figure 2. Explanation of marking objects using intensional navigation

- Basket activities. Dragging and dropping the content of a basket on a class icon causes some operation on the marked objects of the class. A new set of marked objects taken from the basket can replace the existing set of marked

objects, can be summed with it, can be intersected with it, or can be subtracted from it (equivalents to OR, AND and NOT).
- Active extensions. In principle, this capability is introduced to process marked objects rather than to mark objects. However, because all the information on marked objects is accessible from an Active Extension source code the capability can also be used to mark objects.

Intensional navigation and its features allow the user to receive (in many steps but in a simple way) the same effects as through complex, nested queries. Integrating these methods with an extensional navigation, manual selection and other options supports the user even with the power not available in typical query languages.

3.2. Extensional navigation

Extensional navigation takes place inside extensions of classes. Graph's vertices represent objects, and graph's edges represent links. When the user double clicks on a vertex, an appropriate neighborhood (objects and links) is downloaded from the database, which means "growing" of the graph.

Extensional navigation is useful when there are no common rules (or they are hard to define) among required objects. In such a situation the user can start navigation from any related object, and then follow the links. It is possible to use basket for storing temporary objects or to use them as starting points for the navigation.

3.3. Baskets

Baskets are persistent storages of search results. They store object identifiers (OIDs) and sub-baskets. The hierarchy of baskets is useful for information categorization and keeping order. Each basket has its name that is typed in by the user. The user is also not aware OIDs, because special labels are used. During both kinds of navigation it is possible to drag an object (or a set of marked objects) and to drop them onto a basket. The main basket (holding all the OIDs and sub-baskets) is assigned to a particular user. At the end of a user session all baskets are stored in the database.

Aside of creating a new basket, changing its properties or removing items user is able to:

- Perform operations on two baskets: sum of baskets, intersection of baskets, and set-theoretic difference of baskets. The operation result can be stored in one of the participating baskets or in a new one.
- Drag an object and drop it onto an extensional navigation frame. As the result, the neighborhood (other objects and links) of a dropped object will be downloaded from the repository.
- Drag a basket and drop it onto class's visualization in the intensional navigation window. As the result a new set of marked object can be created (replace, add, intersect, subtract). Only objects of that class are considered.

Baskets allow storing selected objects in a very intuitive and structured way. Navigation could be stopped at any time and temporary results (currently selected/marked objects) can be named, stored and accessed at any time.

4. Virtual Schemas

Each data set has at least one virtual schema, which is a direct reflection of the database data. As will be shown in Chapter 5, a single virtual schema has very low resource demand, which means that Mavigator is capable to work with high number of virtual schemas. Every operation on virtual schemas is executed exactly the same way, as it would be on real data. Thus, virtual schemas, in accordance to the views principles, are totally transparent to the user.

Figure 1 shows an example of the virtual schema. All of the examples are based on the well-known Northwind relational database, which is shipped with MS SQL Server. Comparing to the originally relational schema, only names of the associations' roles has been changed. Thanks to that diagram is much more readable.

Generally, Virtual Schemas allow creating customized database views, which consist of the following elements:

- Virtual associations, which are reflections of any dependency among classes. The definition of an association is made up of two roles' names (direct and reverse), two classes' names and two query language statements, which define target set of objects. In the simplest (default) case queries returns all objects from the reverse classes.
- Virtual attributes, which describe objects' properties. Each attribute has name and query language statement, which defines this attribute. In the simplest case it is just the name of a physical attribute.
- Classes, which are counterparts of the physical collections from the database.

Below we will show particular applications of Virtual schemas:

- Determining or changing names of the associations' roles. Because the Northwind schema is based on a relational data source, it utilizes relationships rather than associations. Relationships are identified by primary/foreign keys, and have no names. Such dependencies can be mapped as associations having proper names on their ends. For instance, having CustomerID, we can define the association named OrderedBy between Orders and Customers.
- Creating new connections between classes. For instance we would like to know which companies supply products for a particular order. Instead of navigating through two classes (OrderDetails, Products) we create a new association between Orders and Suppliers. It is quite easy to achieve using path expression of the query language, which defines an essential part of the virtual association.
- More accurate specifying objects from the target class. Let's assume that we would like to analyse only recent information stored in our database. Than we create appropriate virtual schema. Among other information we want to know

only recent orders served by particular employee. In that case, we extend the definition of the Serves role (Employees – Orders) with particular WHERE clause, which check date of the order and returns only objects (ids) with the current date.

- Hiding some classes. It could be useful because of some security reason or just to simplify user's schema. All we have to do is removing particular classes definitions from the virtual schema. Of course, names of the removed classes cannot be used in other definition (i.e. in virtual associations).

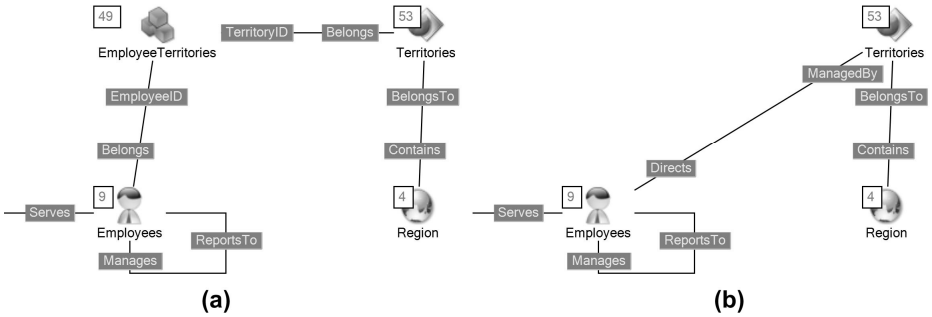

Figure 3. Illustration of hiding intermediary classes.

- Hiding some intermediate classes, which were necessary in relational representation. Figure 3 shows two parts of graph from the Figure 1. Class EmployeeTerritories is a typical intermediate class (with no attributes), which is necessary only to illustrate many-to-many relationship between Employees and Terrotories. Part (a) of the Figure 3 shows the originally graph; part (b) shows the appropriate part without the mentioned class, which improves legibility. That kind of modification requires a few changes to the virtual schema definition: hiding class EmployeeTerritories, and modifying two path expressions describing association.

5. Software architecture and implementation

The presented virtual schemas have been implemented as a part of a data wrapper, which works with the ODRA prototype DBMS. The entire ODRA database management system follows the stack-based approach (SBA) to query languages and uses SBQL (Stack-Based Query Language) [12].

Figure 4 shows the Mavigator's architecture. For this paper the most important are items enumerated below:

- Data source – an ODRA database storing all the information objects,

- Database wrapper – ensures communication, via defined AbstractDatabase2 interface, with any data source. Two things are worth noting:
- All internal data processing regarding information retrieval, works on abstract data representation, which ensures that entire application will be working in the same manner aside of the current data source,

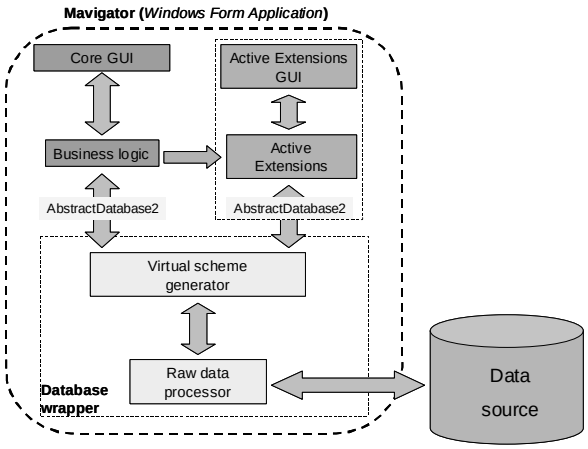

Figure 4. Mavigator's architecture

- Raw Data Processor provides “real” data (in a database native format),
- Virtual scheme generator creates “virtual” data based on the current virtual schema,
- Active Extensions allows adding new functionalities to working application. For more information see [17].

The definition of a virtual schema is stored in an XML file. Figure 5 shows a definition of the new virtual association (Employees - Territories) illustrated on Figure 3 (b). Below we explain the meaning of particular tags:

```
<Association>
   <Name>Directs</Name>
   <ReverseName>ManagedBy</ReverseName>
   <SourceClassName>Employees</SourceClassName>
   <TargetClassName>Territories</TargetClassName>
   <SourceSBQL>Employees.Belongs.EmployeeTerritories.TerritoryID.Territories;</SourceSBQL>
   <TargetSBQL>Territories.Belongs.EmployeeTerritories.EmployeeID.Employees;</TargetSBQL>
 </Association>
```

Figure 5. Definition of a virtual association

- `Name, ReverseName` describes virtual association roles names (direct and reverse),

- `SourceClassName`, `TargetClassName` defines names of the two classes, which will be connected by the association,
- `SourceSBQL`, `TargetSBQL` contain SBQL statements. Generally it could be any valid SBQL query including conditional ones (`WHERE`). However, according to the association semantics, it should return appropriate objects' identifiers (belonging to the "opposite" class). The presented example contains just two path expressions, which simply navigate from one class to another.

6. Conclusions

We have presented the Virtual Schemas developed as a part of visual information retrieval tool Mavigator. Virtual Schemas allow creating customized version of existing database schema. Modifications to the original schema include creating virtual association, virtual attributes, hiding/showing particular classes and renaming items.

We plan to conduct a research on moving a virtual schema from the wrapper level to the application level. Important benefit of such an approach is making virtual schemas wrapper-independent. Potential problems could be connected with performance and ways of cooperation with other wrappers. We also plan to perform some experimental studies with real users regarding the usability of the system.

References

[1] Carey M.J., Haas L.M., Maganty V., Williams J.H.: PESTO: An Integrated Query/Browser for Object Databases. Proc. VLDB (1996) 203-214
[2] Cassel K., Risch T.: An Object-Oriented Multi-Mediator Browser. 2nd International Workshop on User Interfaces to Data Intensive Systems, Zürich, Switzerland, May 31 - June 1, 2001
[3] Fegaras L.: VOODOO: A Visual Object-Oriented Database Language For ODMG OQL. ECOOP Workshop on Object-Oriented Databases 1999, 61-72
[4] Josifovski V., Risch T.: Query Decomposition for a Distributed Object-Oriented Mediator System. Distributed and Parallel Databases J., 11(3), pp 307-336, Kluwer, May 2002.
[5] Kozankiewicz H., Leszczyłowski J., Subieta K.: Updateable XML Views. Proc. of ADBIS'03, Springer LNCS 2798, 2003, 385-399
[6] Murray N., Goble C., Paton N.: Kaleidoscape: A 3D Environment for Querying ODMG Compliant Databases. In Proceedings of Visual Databases 4, L'Aquila, Italy, May 27-29, 1998
[7] Murray N., Paton N.W., Goble C.A., Bryce J.: Kaleidoquery - A Flow-based Visual Language and its Evaluation. Journal of Visual Languages and Computing 11(2), 2000, 151-189

[8] Shipman D.: The functional data model and the data language DAPLEX. ACM Transactions on Database Systems, vol. 6, no. 1, 1981.
[9] Shneiderman, B.: Visual user interfaces for information exploration. In Proceedings of the 54th Annual Meeting of the American Society for Information Science, pages379–384, Medford.NJ, 1991.Learned Information Inc.
[10] Smith M., King P.: The Exploratory Construction Of Database Views. Research Report: BBKCS-02-02, School of Computer Science and Information Systems, Birkbeck College, University of London, 2002
[11] Stolte Ch., Tang D., Hanrahan P.: Polaris: A System for Query, Analysis and Visualization of Multidimensional Relational Databases. IEEE Transactions on Visualization and Computer Graphics, Vol 8, No 1, January-March 2002
[12] Subieta K., Beeri C., Matthes F., Schmidt J. W.: A Stack Based Approach to Query Languages. Proc. of 2nd Intl. East-West Database Workshop, Klagenfurt, Austria, September 1994, Springer Workshops in Computing, 1995.
[13] Subieta K: Mapping Heterogenous Ontologies through Object Views. Proc. of 3-rd Workshop Engineering Federated Information Systems (EFIS 2000), Dublin, IOS Press, 2000, 1-10
[14] Subieta K: Theory and Construction of Object-Oriented Query Languages. Editors of the Polish-Japanese Institute of Information Technology, Warsaw 2004, ISBN 83-89244-29-2, 522 pages (in Polish)
[15] Trzaska M, Subieta K.: The User as Navigator. Proc. of the Eighth East-European Conference on Advances in Databases and Information Systems (ADBIS'2004), pp. 228 - 240, September 22-25, 2004, Budapest, Hungary.
[16] Trzaska M., Subieta K.: Structural Knowledge Graph Navigator for the Icons Prototype. Proc. of the IASTED International Conference on Databases and Applications (DBA 2004)
[17] Trzaska M., Subieta K.: Usability of Visual Information Retrieval Metaphors for Object-Oriented Databases. Proceedings of the On the Move Federated Conferences and Work-shops (DOA, ODBASE, CoopIS, PhD Symposium), Springer Lecture Notes in Computer Science (LNCS 3292), pp. 822-833, October 25-29, 2004, Larnaca, Cyprus.