Mavigator – a Flexible Graphical User Interface to Data Sources

Mariusz Trzaska
Polish-Japanese Institute of IT
Koszykowa 86
Warsaw, Poland
+48 (22) 58 44 564
mtrzaska@pjwstk.edu.pl

Kazimierz Subieta
Institute of Computer Science PAS
Ordona 21
Warsaw, Poland
+48 (22) 58 44 564
subieta@pjwstk.edu.pl

ABSTRACT

We present Mavigator, a prototype of a graphical user interface to databases. The system is dedicated to naive users (computer non-professionals) and allows them to retrieve information from any data source, including object-oriented and XML-oriented databases. The system extends its core functionalities by the Active Extensions (AE) module, which assumes a trade-off between simplicity of user retrieval interfaces and complexity of output formatting functions. In AE the latter are to be done by a programmer using a fully-fledged programming language (currently C#). Thus the retrieved data can be post-processed or presented in any conceivable visual form. Another novel feature of the Mavigator is the Virtual Schemas module, which allows customization of a database schema, in particular, changing some names, adding new associations or hiding some classes.

Categories and Subject Descriptors

H.5.2 [**INFORMATION INTERFACES AND PRESENTATION**]: User Interfaces - Graphical user interfaces (GUI).

General Terms

Management, Design, Experimentation, Human Factors.

Keywords

graphical query interface, information retrieval, information browsing, navigation, baskets, HCI, GUI, database views.

1. INTRODUCTION

The style of working with an application must be designed regarding abilities of the target user. Retrieval capabilities of the visual information retrieval system Mavigator have been designed for naive users, typically computer non-professionals. Such a user cannot deal with sophisticated retrieval methods and metaphors, especially using keyboard-oriented, very high-level languages, such as query languages like SQL and script languages for formatting retrieval output. There are two options: some generic output format (e.g. a table), which is usually too primitive for the users, or some very attractive, user-friendly form (e.g. a function chart), which in turn must be specialized to a very particular application and a retrieval kind. There is a need for tradeoffs between these extremes. Usually such tradeoffs sacrifice the expressive power of the output formatting capabilities.

Mavigator is our second prototype. The first one, called Structural Knowledge Graph Navigator (SKGN) [1], has been designed and developed for the European project ICONS. In Mavigator we assume navigational retrieval and browsing style in formatted data sources a la XML rather than full-text retrieval in raw texts with predefined output format a la Google. In contrast to SKGN, Active Extensions, which are a part of the Mavigator prototype [2], allow extending its existing functionalities by professional programmers, on the order of end users. In contrast to Visage [3], which uses a dedicated script language, Active Extensions are based on a fully-fledged programming language. Due to such solution the extensions do not restrict the form of output, execution speed or algorithmic complexity of output formatting functions.

A disadvantage of our solution is that end users asking for a new output format need cooperation with a professional programmer, who adds the new required functionalities. We believe that this solution is inevitable if we do not want to sacrifice the expressive power of the visual interface. Our experience has shown that in majority of visual retrieval tasks such a mode of making changes to end user interfaces is fully acceptable regarding both the time necessary for the changes and the changes cost.

The Mavigator's retrieval metaphors rely on database schema graphs. Because such graphs can be large and sophisticated, there is a need to restrict them and to customize to particular users. For such purposes, database views could be used. They are subject of research and development for 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. In particular, to the best of our knowledge till now there is no proposal of views that deliver virtual associations (in UML terms) among object classes. Within Mavigator we have implemented such a view mechanism within the module called Virtual Schemas.

The remainder of this paper is organized as follows. In Section 2 we discuss related work. In Section 3 we give an overview of information retrieval capabilities. Section 4 contains information about Active Extensions and Section 5 about Virtual Schemas. Section 6 concludes.

2. RELATED WORK

Related solutions could be analysed from the three points of views: the way of information retrieval, methods of modifying application's functionalities and utilization of database views. However, because of the very limited space we will only give a brief description of the related work. More information can be found in [2], [4].

Roughly, visual metaphors to information retrieval can be subdivided into two groups: based on graphical query languages and graphical browsing interfaces. The subdivision is not fully precise because many systems have features from both groups. An example is Pesto [5] having possibilities to browse through objects from a database. 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.

Typical visual querying systems are Kaleidoscape [6] and VOODOO [7]. 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. We find it very useful and intuitive thus we have adopted it to our metaphor.

The most hard and universal method of modifying application's functionalities is generating a completely new system based on some existing framework. Depending on the designing/developing method, this approach is dedicated for particular groups of users (usually however not naive ones). DRIVE [8] is an example of a user interface to a database development environment. The system dynamically interprets a conceptual object-oriented data language with active constructs. The specification of the interface is made in the textual language called NOODL. The model framework includes the following main class categorisations: user, data, interface, and visualisation classes. Thanks to separation of data and interface, each data item could have associated multiple interface components. Each user could have own set of user-specific views and access privileges. Visual programming facilities help in creating queries, constraints, and other retrieval options. Although DRIVE has been designed as an easy-to-use graphical development system it is disputable if a typical casual user will be able to accept it.

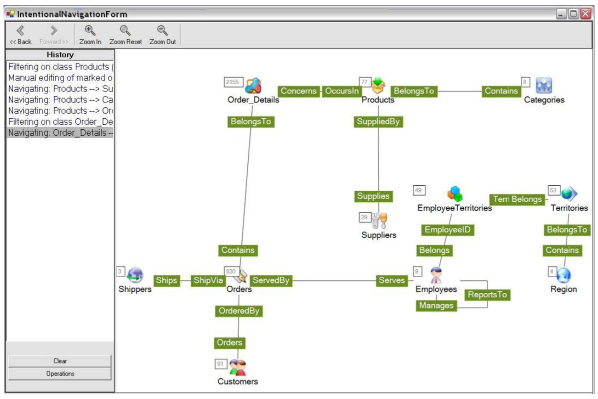
Visage [3] is an example of another approach. The user interface itself contains some navigation methods for retrieval. Moreover, each data visualization component, called frame, could be modified by attaching a special script. Similarly to Mavigator, scripts are written by programmers. In contrast to our approach, Visage utilizes a scripting language similar to Basic. Unfortunately the language interpreter overhead limits the dataset size that can be manipulated with no significant delays. That is one of the reasons for using in Mavigator a fully-fledged programming language.

One of the most advanced examples of views' uses in visual tool is AMOS [9] and its graphical browser GOOVI [10]. This system uses mediators, which are counterparts to views. In contrast to Mavigator, GOOVI browser is an independent tool, which only collaborates with AMOS. In the other words, browser does not have views capabilities, but only works with the system, which supports them.

In the Watson [11] browser the term views is used in a different sense and context. In our terms, their views are some states of extensional navigation, i.e. some number of objects, which are connected by various links. Watson's views have little in common with a database schema and database views.

3. INFORMATION RETRIEVAL CAPABILITIES

Mavigator employs three key metaphors: intensional navigation, extensional navigation and baskets. It allows also for traditional querying via SBQL, an object-oriented query language in the SQL style.


IntentionalNavigationForm

History
Filtering on class Products (
Manual editing of marked o
Navigating: Products --> Su
Navigating: Products --> Ca
Navigating: Products --> Or
Filtering on class Order_De
Navigating: Order_Details --

Figure 1. Mavigator during intensional navigation.

3.1 Intensional Navigation

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

- 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.

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 SBQL predicate based on objects' attributes. The action is analogous to the SQL select clause.
- Manual selection. Using labels it is possible to mark particular objects manually.
- Navigation 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. This activity is similar to using path expressions.

- 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 one, 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.

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 neighbourhood (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 objects (actually objects identifiers) 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. 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 user's 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 set-theoretic operations on two baskets: sum of baskets, intersection of baskets and difference of baskets.
- Drag an object and drop it onto an extensional navigation frame. As the result, the neighbourhood 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 through operations such as replace, add, intersect and 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. ACTIVE EXTENSIONS

Active Extensions of the Mavigator are created using a programming language. We have assumed, however, that a Mavigator's user is not a programmer and will not be able to create such extensions. Hence some professional programmer must come into the play.

Mavigator already employs some information retrieval metaphors, which are powerful and yet easy-to-use, so we have decided to provide a way to add new functionalities operating only on a query result. The approach does not complicate the entire application's architecture and guarantees sufficient flexibility.

The current prototype uses Microsoft C# as a language for active extensions. A programmer is aware of the Mavigator metadata environment, which allows him/her to write a source code of the required functionality in C#. Writing of the Active Extension source code is done in Mavigator's special editor. Once programmer compiles the code, particular Active Extension is ready to use (without stopping Mavigator). Then the end user is supported with one click button causing execution of the written code. The functionality of such programs is unlimited. Next paragraphs present its particular applications.

A simplest type of Active Extensions is a calculation based on a retrieval result. In Mavigator we have implemented the most popular aggregate functions: a minimal attribute's value, a maximal attribute's value and an average attribute's value. They are very easy in use. The users have to select a particular type of calculation and a particular attribute in the query result. Then, the calculation result is shown.

Another application of Active Extensions are active projections, which allow visualizing a set of objects where position (x and y coordinates) of each of them is based on value of particular objects' attributes. Current implementation uses two axes (2D), which allow visualizing dependencies of two attributes. Active projections make it possible quick identification or data mining concerning groups of objects having some properties. Besides the visual analysis of objects dependencies it is also possible to utilize projections in more active fashion. Object taken from a result retrieval basket can be dropped on projection's surface, which cause right (based on attributes values) placement. It is also possible to perform reverse action: drag an object from the surface onto the basket (which cause recording object in a basket).

The last sample application of AE is objects exporter, which allows cooperating with other software systems. It is possible to send a query result to other programs, such as Excel, Crystal Reports, etc. This feature makes it possible subsequent sophisticated processing of Mavigator's results of querying/browsing. The current prototype exports query results to XML files, which could be post-processed by many modern applications.

5. VIRTUAL SCHEMAS

A virtual schema (a view) allows for customization of a database schema according to needs of the current user. A virtual schema exists as a definition only, no physical mapping of data is performed. According to the fundamental transparency requirement, the user uses a virtual schema in the same manner as an original database schema. Virtual schemata have low resource demands, thus Mavigator is capable to support user's work with many virtual schemas in the same retrieval session.

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

- Virtual associations, which reflect any dependencies among classes. The definition of an association is made up of two roles' names (direct and reverse), two classes' names and two queries, which define target sets of objects.
- Virtual attributes, which describe objects' properties. Definition of a virtual attribute consists of a name and a query. In the simplest case it is just the name of a physical attribute.
- Classes, which are counterparts of physical collections from the database.

Particular applications of Virtual Schemas are the following:

- Determining or changing names of the associations' roles.
- Creating new connections between classes. It is quite easy to achieve that by using path expressions, which define essential parts of virtual associations.
- More accurate specifying objects from the target class. Let's assume that we would like to analyse only recent information stored in our database. In an 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 checks the date of the order and returns only objects (ids) with the current date.
- Hiding some classes. It could be useful for security or just to simplify user's schema.
- Hiding some intermediate classes, which were necessary in a relational representation. Mavigator could be connected (via a dedicated wrapper) to any data source. Thus in case of a relational one, it would be useful to hide intermediary tables (mapped as "empty" classes), which were necessary only to illustrate many-to-many relationship.

6. CONCLUSIONS AND FUTURE WORK

We have presented Mavigator, which offers new quality in two main areas. The first one is extending existing application's functionalities by Active Extensions, which use fully-fledged programming language and make it possible to create any kind of additions to Mavigator's core functions. The second area contains Virtual Schemas, which allow creating customized version of the existing database schema. Possible modifications to the original schema include creating virtual association, virtual attributes, hiding/showing particular classes, etc.

The architecture of the Mavigator is flexible and allows one to work with any kind of data source. The utilized data retrieval metaphors are easy to understand even for casual users.

As a continuation of our research, we have started a work related with connecting the Mavigator with the eGov-Bus virtual repository. This data source is developed as a part of the EC project called eGov-Bus[1].

7. REFERENCES

[1] Trzaska M., Subieta K.: Usability of Visual Information Retrieval Metaphors for Object-Oriented Databases. Proceedings of the On The Move Federated Conferences and Workshops (DOA, ODBASE, CoopIS, PhD Symposium), Springer Lecture Notes in Computer Science (LNCS 3292), pp. 822-833, October 25-29, 2004, Larnaca, Cyprus.

[2] Trzaska M., Subieta K.: Active Extensions in a Visual Interface to Databases, Fourteenth International Conference on Information Systems Development (ISD´2005), Kluwer/Plenum Press, 14-17 August, 2005, Karlstad, Sweden.

[3] Roth F., Chuah M., Kerpedjiev S., Kolojejchick J., Lucas P.: Towards an Information Visualization Workspace: Combining Multiple Means of Expression. Human-Computer Interaction Journal, Volume 12, Numbers 1 & 2, 1997, 131-185.

[4] Trzaska M.: Virtual Schemas in Visual Interfaces to Databases, I Krajowa Konferencja Naukowa "Technologie Przetwarzania Danych", pp. 361 - 371, 26-28 September, 2005, Poznan, Poland.

[5] Carey M.J., Haas L.M., Maganty V., Williams J.H.: PESTO: An Integrated Query/Browser for Object Databases. Proc. VLDB (1996) 203-214

[6] Murray N., Goble C., Paton N.: Kaleidoscape: A 3D Environment for Querying ODMG Compliant Databases. In Pro. of Visual Databases 4, L'Aquila, Italy, May 27-29, 1998

[7] Fegaras L.: VOODOO: A Visual Object-Oriented Database Language For ODMG OQL. ECOOP Workshop on Object-Oriented Databases 1999, 61-72

[8] Mitchell K., Kennedy J.: DRIVE: An environment for the organised construction of user interfaces to databases, 3rd International Workshop on Interfaces to Databases, Springer-Verlag Electronic WIC (1996).

[9] 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.

[10] 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

[11] 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.

[1] Advanced Government Information Service Bus (eGov-Bus, IST 26727 STP) is a project supported by the EC as a part of the Sixth Framework Programme.