ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Jessie Kennedy and Peter Barclay (Eds)

Interfaces to Databases (IDS-3)

Proceedings of the 3rd International Workshop on Interfaces to
Databases, Napier University, Edinburgh, 8-10 July 1996

Paper:

Desktop Objects: Directly Manipulating Data and Meta Data

Armstrong Kadyamatimba, John A. Mariani and Peter Sawyer

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

Desktop Objects : Directly Manipulating Data and Meta Data

Armstrong Kadyamatimba, John A. Mariani¹ and Peter Sawyer

Computing Department
Lancaster University
Lancaster
LA1 4YR

{arms, jam, sawyer}@comp.lancs.ac.uk
tel : Phone: +44 1524 593797

Abstract.

In this paper, we discuss the potential for the application of the popular and well-known desktop metaphor to object-oriented databases (OODBs). We describe an initial prototype, the Oggetto Desktop, which supports browsing of both the type and structural lattices of an OODB and direct manipulation for accomplishing schema evolution.

KEYWORDS: object-oriented databases, HCI for databases, the desktop metaphor, direct manipulation.

1. Introduction

The emergence of object-oriented database systems (OODBs) towards the end of the 1980s coincided with the widespread availability of low-cost graphical workstations. OODB development was spurred by the requirements of new application domains for databases which required the modelling of entities whose complexity and heterogeneity posed insuperable difficulties for existing database systems. These two facts have stimulated work on graphical OODB user interfaces to enable users to readily distinguish between types, navigate the type lattice and interact with instances (e.g. [1, 10, 16]). As observed by Almarode [2]:

"With the increased complexity of inheritance, behaviour, and complex objects in the database, building such tools for an object-oriented database management system will provide a challenge for all researchers and implementors"

Oggetto [12] is an OODB which has been the subject of a number of different experimental OODB user interfaces. To date these have concentrated on the integration of user interface components with the database schema (Moggetto [18]) and the use of 3-dimensional graphics to enhance the visualisation of relationships between data, and users' awareness (Q-PIT [4]). In this paper we present a further experiment with OODB user interfaces using Oggetto; the adoption of the desktop metaphor [9] for browsing and querying.

The motivation for this work is derived from the following observations:

- Graphical user interface usability is often enhanced if a coherent metaphor [11], representing an analogy with users' real-world experience of the application domain, is adopted.
- Suitable metaphors for applications of an OODB may be derivable on a per-application basis. However, we are interested in generic OODB user interfaces which will serve for any application domain. Cards and card indexes have been successfully applied to simpler database models (e.g. FileMaker™Pro [7]) but are less satisfactory for complex, heterogeneous objects. Electronic forms have been proposed as an appropriate generic metaphor for the direct manipulation of individual objects [5, 15] but there is no obvious candidate for a unifying metaphor for interaction with an OODB at the schema level.
- The desktop metaphor is widely used and accepted for graphical user interfaces to operating systems - especially the manipulation of file stores and devices, and the invocation of application programs.

In the absence of a direct real-world analogue, we have applied the ubiquitous electronic desktop to Oggetto in the expectation that it would meet our requirements for a generic user interface metaphor and serve as a coherent cognitive aid for users. Although the work is still in progress, we have discovered that the metaphor is applicable with surprisingly little adaptation or grafting of artificial concepts.

The rest of the paper is organised as follows : in the next section, we examine both the desktop metaphor and OODBs, looking at existing browsing interfaces, concluding with a small set of requirements for an OODB desktop.

¹ The author to whom all correspondence should be addressed

In section three we introduce the Oggetto Desktop, a prototype system which embodies some of these requirements. The use of direct manipulation for schema evolution is presented in section four. The metaphor and the Oggetto Desktop leads us to a number of possible extensions to the system : incorporating querying as part of the functionality of the desktop and allowing access to object methods through the desktop. These possibilities are discussed in section five. In the final section, we present our initial conclusions on the desktop metaphor's suitability as an interface to OODBs.

2. OODBs and the desktop metaphor

User interface metaphors aim to enhance computer systems' ease of use by providing interactive graphical analogues of users' physical working domains. The desktop metaphor is undoubtedly the most pervasive and successful metaphor. Here, data files, programs and devices are represented as graphical entities which simulate objects which may be found in a user's physical office; folders, paper files, rubbish bins, mail boxes, etc. Events generated by users' *direct manipulation* [19] of these are mapped onto operations on the underlying entities. The convention used to represent entities as graphical analogues of real-office objects helps users to bridge the *gulf of evaluation*. Similarly, the fact that entities' "types" are used to conceal from the user the need to know which application to invoke or the number and type of arguments etc., helps to bridge the *gulf of execution* [13].

Of course, metaphors have limits. It would be over-constraining if a user interface only supported those functions or operations for which a physical-world analogue existed. However, it is perhaps a mark of a good user interface if functionality with no or tenuous, physical analogues can be integrated with a metaphor without over-complicating the user's mental model of the underlying application. Users of the Macintosh desktop, for example, generally cope with computer-specific functions (e.g. ejecting and erasing disks) after a little practice. This has led us to investigate the desktop metaphor to discover whether OODB browsing and querying could be seamlessly integrated into an electronic desktop.

The normal way to develop a metaphor-based graphical user interface is to analyse the application and build an electronic analogue of its physical-world counterpart. An OODB lacks an obvious real-world counterpart so we have started by examining existing user interfaces for other application domains and asking whether any of these could be adapted to our requirements. Having settled on the electronic desktop as the most promising, our hope is that users will be sufficiently familiar with it to offset the abstract nature of the underlying application, enabling them to interact with an OODB as if they had physical world experience of objects, types, queries, etc. to guide their interactions.

It is useful at this point to review our requirements of an OODB user interface. Our first requirement is for a means for searching the object space.

The relational database model enabled ad-hoc, declarative querying as a means of searching the database. This is straightforward for the user to accomplish where there are a few relational tables and by-example [14] tabular user interfaces. With OODBs, by contrast, while querying *is* useful, the heterogeneity of object types means that a navigational style of searching (*browsing*) is often more useful. There are frequently so many object types, with such complex internal structure (e.g. containing of components of both primitive and object types) that formulating a query is difficult because of the volume of this information required *a-priori* by the user in order to do so. Hence the user requires a means to browse the database; to navigate around the database following relationships between entities.

Browsing of an OODB can be accomplished on two dimensions; browsing the type lattice (following specialisation and generalisation relationships) and the structural lattice (following composition relationships).

Figure 1, taken from [18], illustrates the type lattice (rear-most window) of an Oggetto database. Each node represents a type; nodes connected to the right of a node are subtypes (specialisations); nodes connected to the left are supertypes (generalisations). For example, "Peripheral" and "Computer" are subtypes of "Device" which is a subtype of "Object". One scenario for browsing the database may be to traverse these specialisation relationships to search for a given type and, when found, all the instances of that type may be listed (e.g. the instances of the "SE30" type are listed in figure 1).

Figure 1 An Oggetto type lattice

This approach is similar to that taken by O2 with O2Look and O2Tools; see the examples in [6]. The type lattice may be represented graphically, permitting the user to rapidly assimilate the specialisation/generalisation relationships in the database. However, in anything other than a small-scale application, the type lattice rapidly grows to the point where it cannot be displayed in a form which is easily assimilated. Various solutions have been developed to address this problem, of varying effectiveness. For example: zoom in/out, concealing of branches or levels of the lattice, etc.

Cutting across the type lattice is the structural lattice. This defines how instances of types (objects) are composed of other objects. For example, in figure 1, the type "SE30" contains two attributes, "peripherals" and "network", whose values are of the object types "Peripheral" and "Network" respectively. Contrast these with the attribute "memory" whose value has a primitive type (integer). Conceptually an SE30 is a composite of instances of primitive types and instances of "Peripheral" and "Network". In fact, the values of "peripherals" and "network" are actually references to the corresponding instances of "Peripheral" and "Network" and hence may be traversed to explore the structure and state of an "SE30" object. Consider a more complex example, where an object of type "car" might be composed of objects of type "engine", "transmission_system", "electrical_system" and so on, each themselves decomposing to many lower levels. A "car" object may actually be a composite of thousands of others.

Support for these two browsing dimensions is essential if the desktop metaphor is to be applicable to OODBs. It must cope with large type lattices and very large composite objects.

So far we have considered making users' manipulation of an OODB fit within the general set of rules defined by the desktop metaphor. We have done this by drawing analogies between OODB entities and operations, and the established application domain of the electronic desktop; file stores and operating systems. However, there are some OODB-specific activities which do not have such a ready set of analogues. One of these is the question of schema evolution; the manipulation of the type lattice and type definitions. There are of course many difficult problems here (see [3]) but as Oggetto supports a subset of schema manipulation operations, we would wish the desktop metaphor to at least provide a user interface to these which cleanly dovetailed with the other type and object operations.

In summary, an OODB electronic desktop should:

- Provide mechanisms to browse the type lattice.
- Provide mechanisms to browse the structural lattice.
- Cleanly integrate querying operations with the above browsing mechanisms.
- Provide consistent, transparent mechanisms to manipulate and invoke applications and object methods.
- Support other OODB-specific operations such as schema evolution.

This section has outlined the initial premise upon which our adoption of the desktop metaphor has been based and identified some requirements of the desktop metaphor if it is to adequately support a user interface to an OODB. In the following sections we develop this theme to illustrate how the ideas have been implemented and illustrate some issues raised.

3. Browsing the desktop

The desktop metaphor reduces to a WIMP style interface; we have windows, icons, menus and pointers. Icons are typically used to represent single items (i.e. a file or document) and groups of items (i.e. a folder or directory). To generalise these terms away from "files" and "folders", we introduce the following terms :

- single item : an entity such as a file (in the Macintosh style) or an object (in a database).
- single icon : the icon representation of a single item.
- group item : an entity which represents a group of collection of items (single and/or group). In Macintosh terms, a folder.
- group icon : the iconic representation of a group item.

By clicking on a single icon, we open (access) the underlying information. By clicking on a group icon, we open that icon into a window, which in turn contains further single/group icons. We are all familiar with this style of interface. The group items form a hierarchy, through which we can navigate by repeatedly opening group items.

Figure 2 illustrates a simple example of an Oggetto database used in the examples below. There are eight object types organised hierarchically from root (the most general) to wife (the most specialised). The ellipses represent instances of the types to which they are connected.

Figure 2 : Example Type and Instance Hierarchy

3.1. Browsing the type lattice

When the user activates the Oggetto Desktop, they are presented with the following display (figure 3). As can be seen, there is a group icon called "root" which forms the top view of the database.

Figure 3 : The Initial Desktop

When the user clicks on the "root" group icon, it opens into the window shown in figure 4.

Figure 4 : The opened root group item

This window contains all the immediate root subtypes in this instantiation of an Oggetto database. We can continue browsing the schema of the database by selecting the group icons which represent types. For example, by selecting the "vehicle" group icon, we obtain the window shown in figure 5.

Figure 5 : The "vehicle" window

We can continue opening group icons at any level of the type lattice, allowing us to move up and down through the structure.

3.2. Browsing the instance structure

As soon as we arrive at a type which has instances, those instances are represented by single icons. In figure 6, we have selected the "person" group icon, resulting in the "person" window. This window contains the further group icon, "married_person" (a sub-type of person) and three single icons representing three objects of type "person", called PAUL, CLAIRE and EMMA.

Figure 6 : type "person" showing subtype and three instances

Similarly, figure 7 shows the results of selecting the "married_person" group icon, resulting again in a single subtype (wife) and three instances of "married_person".

Figure 7 : the contents of "married_person"

Our adaptation of the desktop metaphor simply treats object types as folders (group icons) which contain both other types (subtypes) and instances. It is unusual to find schema and instance information mixed together in a single representation of a database - in figure 1, for example, only the schema is displayed in the type hierarchy window and a display of instances has to be explicitly requested by selecting a type node. Representing an Oggetto database as an electronic desktop, however, means that every object type expands into a window, easing one of the usual obstacles to mixing the display of both types and objects; lack of space. Simply having sufficient screen space is not enough, of course. The user should be able to easily distinguish between instances and types. This is achieved simply by the use of distinct visualisations (single and group icons) and ordering the items so that all types are listed first, and instances occupy the remaining space.

Another feature of this application of the desktop metaphor is that it reinforces the fact that in OODBs such as Oggetto, there is no rigid distinction between data and meta-data. A consequence of this is that, where appropriate, the meta-data may be manipulable as if it were data. This is discussed in section 4.

If a type is conceptually a container of subtypes and instances, an instance may also be considered a container of instances of primitive and object types (c.f. the type SE30 in figure 1). We therefore follow the same mechanisms for displaying types through to the display of objects; if we select a single icon, it too expands into a window.

In figure 8, we show the results of selecting the DAVID icon from the window shown in figure 7. An Oggetto object may contain attributes or references to other objects. Any simple single valued attributes are presented as label and value pairs at the top of the window. In our example, we have the five simple attributes ("age", "face", "name", "quote", and "sex"). Multi-valued simple attributes will be presented as pop-up menus. Any object references are presented at the bottom of the window as single or group icons. In figure 8, we have the group icon "has_child" and the single icon "married_to".

The Desktop user can thus, from initially browsing the type lattice, encounter object instances which can be further browsed to explore the structure of the stored data, without switching metaphors or indeed contexts.

Figure 8 : viewing an instance which contains simple valued attributes, an object reference, and a further collection of objects

The "visualisation" of an instance maintains the desktop metaphor, by presenting any multivalued attribute consisting of object references as a further group icon. Similarly, any single valued attribute consisting of an object reference is represented as an icon. In this case, we have a group icon called "has_child" (the name of the attribute). By opening this icon, we will display the three object instances (as icons) for PAUL, EMMA and CLAIRE. By opening the object icon labelled "married_to" on the right of the display, we will display the object instance RUTH (in a similar manner to that of DAVID).

The major difference between our approach and that of standard browsing techniques is that it is possible to mix browsing of the type and structural lattices. The initial way into the data is via the root of the type lattice. However, at whatever point in the type lattice that we have concrete types, we immediately present and allow access to instances of these types. It is thus possible to easily switch from type to instance browsing, within the same metaphor.

Similarly, techniques used in browsing subtypes (the presentation and selection of group icons) are also applied when browsing structured instances; in our simple example, moving from the single instance DAVID to the three objects representing his children.

3.3. Manipulating Object Instances

Users may directly manipulate objects. For example, when deleting objects, it is possible to use the "drag and drop" approach; we can select an object and drag it to the Desktop wastebasket.

When creating a new object, we can use a combination of selection and menu usage. By selecting either a group icon representing a type or the open window of a type, followed by the desktop menu operation "new object", we cause a new unnamed single icon to appear in the type window.

By then selecting this single icon and opening the window we can then interact with the currently empty attribute. For example, if we were creating the RUTH object, we would have an "empty" window as shown in figure 9.

Figure 9 : an "empty" object instance

The simple attributes can be filled in by selecting the slots and typing in values directly. The single icons (married_to) can be replaced either by renaming the null icon or by dragging the DAVID icon (if visible) from elsewhere on the desktop over the null icon. Similarly, the "has_child" group icon can be replaced by dragging the "has_child" icon from the open DAVID window (if visible). Otherwise, the individual icons for PAUL, CLAIRE and EMMA can be selected from elsewhere and dragged onto the group icon.

4. Desktop schema evolution

We believe that the desktop metaphor, with its attractive features of direct manipulation, offers a basis for a high-level interface to the problems of schema evolution. The Oggetto system offers a set of primitives which can be applied to the process of evolution; the challenge for the desktop metaphor is how it can be used to specify the desired activities. Rather than exhaustively detail the evolution operations supported by Oggetto and their realisation in the Desktop, we focus on a worked example, showing how it can be achieved in OQL (Oggetto's Query Language) and the Desktop.

The portion of the database we are interested in contains the type Person, and the following five instances : DAVID, RUTH, CLAIRE, EMMA and PAUL. The actions we wish to undertake are as follows :

1. Introduce a new type, married_person, with attributes spouse and children of type Person and make it a subtype of Person
2. Change the types of the instances DAVID and RUTH from Person to married_person
3. Set the values of the new married_person attributes of DAVID and RUTH to the appropriate values.
4. Realise that spouse should have type married_person, and, change the type of the spouse attribute in married_person from person to married person.

To do this using OQL, we undertake the following operations.

1.

```
type married_person (
    person spouse; person has_child
) inherits person;
```
2.

```
schema change type of DAVID to married_person
schema change type of RUTH to married_person
```
3.

```
DAVID -> spouse := RUTH;
DAVID -> has_child := (CLAIRE, PAUL, EMMA);
RUTH -> spouse := DAVID;
RUTH -> has_child := (CLAIRE, PAUL, EMMA);
```
4.

```
schema change attribute type spouse in married_person to married_person
```

To do this using the Desktop :

1. In the Person window, select the "new subtype" menu option. Interact with the form window to introduce the new attributes.
2. Select the DAVID and RUTH objects from the Person window, and drag them to the married_person folder or window, and drop them there.
3. We will consider the DAVID object. Open the DAVID object. Make copies of the RUTH, EMMA, PAUL and CLAIRE objects. Select the copies and drag them to the appropriate slots / folders in the DAVID window.
4. Open the married_person window. Select the type of the spouse attribute. Change its type by overtyping.

5. Future Work

In this section, we present some areas we wish to explore using the Desktop.

5.1. Desktop methods

A conventional desktop metaphor is essentially a veneer over a window management system through which an operating system's file system and functions are manipulated. One way to view this is that the point at which individual entities (tools, files, application programs) are opened is the point at which the desktop should relinquish control to the underlying application/tool/etc. The same may be true of an OODB desktop; the point at which individual objects are displayed is the point at which the desktop ceases to be in control; at least within the physical boundaries of the object's user interface.

As currently implemented, the Oggetto desktop simply permits users to interact with objects by viewing them as windows containing textual and iconic representations of their properties. This has the advantage of following the desktop metaphor right through from schema browsing to browsing of the objects' structural lattices. However, this may be viewed as a default mechanism for displaying objects. There is a case for the desktop relinquishing control where objects have specialised, type-specific display mechanisms already defined.

Oggetto's database programming language is not computationally complete and methods are restricted to derived-value methods which are evaluated when an object is accessed and displayed as ordinary attributes. However, an OODB where more complete functionality, is encapsulated by object methods could use the desktop metaphor simply as the means to invoke a "display" method. Here, when an object's icon is clicked upon, its own "display" method may be invoked to display it instead of having every object displayed using the same mechanism. Hence every object type may have its own unique user interface; "person"s as digitised photographs, "car"s as 3-dimensional schematics, etc. What is important is that objects having different display methods should not imply extra work for the user in order to invoke those display methods. A keystone of effective user interface design is consistency and this can be achieved by the desktop manager mapping a single, recognised event (in this case, double-clicking on a single icon) onto the appropriate display method.

Other object methods which should be invocable from the desktop are those which affect the state of the desktop itself such as creation (e.g. as a "new" menu option) and deletion (e.g. by dragging an object to the waste basket) methods. This small set of methods should be, indeed are, common to all objects (in terms of message signatures, although the functionality of the invoked methods may vary). Hence, in our proposed desktop, every

object type would be able to redefine the default "create", "delete" and "display" methods and the desktop manager would map these to common input events to enforce consistency.

5.2. Querying the desktop

Browsing and querying are very different approaches to retrieving information. With OODBs, both means of retrieval are required. The problem arises of how to integrate the two techniques without compromising the cohesiveness of the metaphor. In the case of the desktop applied here, existing applications of the desktop metaphor already provide the answer. The Apple Macintosh menu bar supports a "find" option under the "File" entry. When this is employed, folders containing matching file/folder names are opened and brought to the desktop. By simple extension, a similar mechanisms can be employed to initiate a query of the Oggetto OODB.

The Macintosh find option can only be thought of as an extremely rudimentary search of the contents of a Macintosh disk. We are only allowed to specify a single attribute; the name of the file/folder. In practice, the authors have often wished for something slightly more powerful; for example, if we know the type of the file (e.g. MacDraw) we are after instead of or as well as its name.

Clearly, the simple single attribute retrieval is not powerful enough for OODB purposes. Instead, we propose to use an object-oriented by-example [14] query tool of the kind described in [17] where the query template form is dictated by the currently selected type (indicated by the currently selected folder (type) or icon (instance)).

The results of the query could be presented again in manner analogous to that of the Macintosh desktop; by the opening and presenting of types and objects. The user could step through the answer set by using the "again" menu entry. The types and objects thus displayed forming browsing continuation points.

Although more complex and powerful than the Macintosh find option, we believe that fundamentally it is the same and would dovetail with the desktop as a pull-down menu function equally effectively.

In addition, by borrowing concepts from visual programming languages, it may be possible to use direct manipulation to specify queries (using an open instance window as the basis for a query "form"), perhaps even join type operations (by overlapping group icons representing types).

An alternative which we currently favour is the provision of query folders. A query folder will have an associated query, which can be input either as an OQL select statement or via a form-filling interface. Once specified, a query folder will reside in the type folder of the type it operates on. Thereafter, when a user selects a query folder, the associated query will be executed, and the folder open into a window containing the results of the query. By employing a specialist where the query statement is complex, end-users have a simple mechanism for calling up the results of a complex query.

Direct manipulation of the folder representations of queries will allow a visual programming language to be developed which will allow the specification of union, intersection and difference of query results. This will allow users to build quite complex queries from seemingly simple visual building blocks.

5.3. Scaling Up

The most important open question about the use of the desktop metaphor, however, is that of whether it will scale up. We feel that allowing the user to browse by successively opening and closing windows representing types and instances is an approach which users will naturally adapt to and allow them to manage their display space. However, we have yet to experiment with really large data bases.

In situations where a type has hundreds or thousands of instances, we believe a Starfield [8] approach could prove useful, where the layout of "micro-icons" is dictated by a 2D scatterplot. Starfield interaction techniques for zoom and bloom could be integrated within the desktop metaphor.

Intuitively, we suspect that, as with existing applications of the desktop metaphor, users will cope provided that the type hierarchy is neither too deep or too broad. For example, if types frequently have tens of immediate subtypes, users will be cognitively overloaded. Multiple inheritance may also raise particular problems. We plan to explore these issues more fully when our current prototype implementation is complete.

A further possibility is to have a threshold for how many objects a user is willing to have displayed. If the window would contain more than 10 (say) objects, an instance folder could be displayed with an indication of its size.

The query folders proposed in the previous section will provide an effective filtering mechanism. It will be interesting to find out just how much manipulation of data and meta-data is possible through the use of instance and query folders without having to frequently examine individual instances. However, we are concerned about overloading the folder concept.

6. Conclusions

In this paper, we have suggested that the well-known desktop metaphor can usefully be applied to browsing of object-oriented databases. Moreover, the direct manipulation features of the desktop can be powerfully employed to support population and manipulation of data instances. It also holds substantial promise for the graphical support of schema evolution activities.

We have presented an initial prototype for the Oggetto desktop, and illustrated its use in browsing both type and structural lattices for a simple database and in schema manipulation. We have outlined some issues and areas we intend to explore further.

Just as we believe our interface inherits all the advantages of the desktop metaphor, it must also take the bad with the good. For example, there is a danger of cluttering up the screen with too many open windows. We are unsure as to whether showing classes and instances together is an advantage or not. It is too early in our work to make conclusive statements on these issues, but they will certainly be examined during our usability tests of the system.

7. References

1. Agrawal, R., Gehani, N., Srinivisan, J.: OdeView: The Graphical Interface to Ode, In: H. Gracia-Molina and H. Jagadish (Eds) *Proc. of the ACM SIGMOD International Conference on Management of Data*, ACM Press, 1990.
2. Almarode, J.: Issues in the Design and Implementation of a Schema Designer for an OODBMS. *Proc. ECOOP '91*, Geneva, July 1991.
3. Banerjee, J., Chou, H-T, Garza, J., Kim, W., Woelk, D., Ballou, N.: Data Model Issues for Object-Oriented Applications, *ACM Transactions on Office Information Systems*, 5 (1), 1987.
4. Benford, S., Mariani, J.: Virtual Environments for Data Sharing and Visualisation - Populated Information Terrains, In P. Sawyer (ed) *Interfaces to Database Systems*, Springer-Verlag, 1994.
5. Collet, C., Brunel, E.: Definition and Manipulation of Forms with FO2, In: E. Knuth and L. Wegner (Eds) *Visual Database Systems II*, North-Holland, 1992.
6. Deux, O., et al "The O2 System", *Communications of the ACM*, Vol. 34, No. 10, October 1991, pp 34-48
7. FileMaker™Pro: FileMaker™Pro, Claris Corporation, 1992.
8. Jog, N. and Shneiderman, B., "Starfield information visualisation with interactive smooth zooming", in *Proc. 3rd IFIP 2.6 Working Conf. Visual Database Systems* (Lausanne, Switzerland), 1995
9. Johnson, J., Roberts, T., Verplank, W., et al.: The Xerox Star: A Retrospective, *IEEE Computer*, 22 (9), 1989.
10. King, R., Noval, M.: Designing Database Interfaces with DBface, *ACM Transactions on Information Systems*, 11 (2), 1993.
11. Marcus, A.: Human Communications Issues in Advanced UIs, *Communications of the ACM*, 36 (4), 1993.
12. Mariani, J.: Oggetto: An Object Oriented Database Layered on a Triple Store, *The Computer Journal*, 35 (2), 1992.
13. Norman, D.: Cognitive Engineering, in: D. Norman and S Draper (eds) *User Centred System Design*, Lawrence Erlbaum Associates, 1986.
14. Özsoyoglu, G., Wang, H.: Example-Based Graphical Database Query Languages, *IEEE Computer*, 26 (5), 1993.
15. Sawyer, P., Sommerville, I.: Direct Manipulation of an Object Store, *Software Engineering Journal*, 3 (6), 1988.
16. Sawyer, P., Colebourne, A., Sommerville, I., Mariani, J.: Object-Oriented Database Systems: A Framework for User Interface Development, In R. Cooper (ed) *Interfaces to Database Systems*, Springer-Verlag, 1992.
17. Sawyer, P., Sommerville, I.: MGA: Rule-Based Specification of Active Object-Oriented Database Applications, *Information and Software Technology*, 37, (4), April 1995. pp 203 - 211

18. Sawyer P., Colebourne A., Mariani J., Sommerville, I.: "Database object display definition and management with Moggetto", Proc. 3rd IFIP Working Conference on Visual Database Systems, Lausanne, Switzerland, March 1995. pp 259 - 272.
19. Schneiderman, B.: Direct manipulation: a step beyond programming languages, *IEEE Computer*, 16 (8), 1983.