SUPER - Visual Interaction with an Object-based ER Model

Annamaria Auddino, Yves Dennebouy, Yann Dupont, Edi Fontana,
Stefano Spaccapietra and Zahir Tari

Ecole Polytechnique Fédérale - DI - Laboratoire Bases de Données
IN - Ecublens 1015 Lausanne Switzerland
auddino@elma.epfl.ch

Abstract

SUPER is a project aiming at the specification and development of a consistent set of visual user interfaces covering all phases of the database lifecycle.
In this paper we discuss the basic principles which, in our opinion, should underline a global approach to visual interaction with advanced data models. Visual interaction in SUPER environment is based on direct manipulation of objects and functions, providing users with maximum flexibility during schema definition as well as query formulation. Graphical interactions are easy to manage, and take advantage of the support of a simple but powerful modelling paradigm. Visual data manipulation is assertional and object-based. The environment offers multiple interaction styles, well-suited for various categories of users. Interaction styles are consistent over the various functions and editors.
To support the discussion, SUPER schema and query editors are analyzed, focusing on functionalities, and the underlying design choices, rather than precisely describing how they operate. An example of query formulation shows the rules used to govern interactions with users.

1 Introduction

Visual interaction had a drastical evolution during the eighties. WYSIWYG techniques (What You See Is What You Get) are nowadays standard for personal computing, while the WIMP metaphor (Windows, Icons, Menus, Pointing devices) governs user interaction with larger systems on workstations (and is moving into personal computing as well). Consequently, researchers try to master the many existing possibilities for human computer interaction. User Interface Management Systems (UIMS) are becoming popular as an answer to this question.
Despite this evolution, users of database management systems are still bound to classical textual languages, namely SQL. Although proposals for visual languages have been well known at least since 1975, thanks to QBE [37], research on visual interfaces still has to produce a global, recognized framework, consistent with the actual state of the art in data modelling techniques.
Indeed, graphical data definition techniques is the only area where a large consensus has been achieved on the marketplace. A number of tools exist, which offer graphical facilities for the definition of a database schema, according to concepts of the entity-relationship (ER) approach. They are eventually complemented with an automatic translation into a relational schema. Some tools also provide functionalities for describing application processes (usually through dataflow diagrams, sometimes with

Petri nets) and consistency checks. As far as we know, no commercial tool proposes a graphical manipulation language.
Several prototypes providing graphical DBMS interfaces have also been developed. Some of them only support graphical data definition: [8], [2]. DDEW [26] extends the definition process to all phases of database design, providing an integrated environment from user requirements to physical design. Some other tools provide both schema definition and visual querying facilities: ISIS [13], SNAP [6], [28], Pasta-3 [18, 19].
A few prototypes support visual data browsing, rather than query formulation: [21], ZOO [27], OdeView [1]. Finally, some prototypes only provide an aid for query formulation, to relieve users from constraints of textual syntax: [20], for instance, uses syntax graphs to guide users through the formulation of a relational query. Outside the scope of this presentation are toolkits for the design of graphical DBMS interfaces, like FaceKit [17], which belong to research in UIMS.
Existing prototypes can be classified according to the underlying data model (the following list of prototypes is not meant to be exhaustive):

- Entity-Relationship model: [8], [35], [36], [10], [21], [28], [26], [7], [14],[18], [9], [19];
- Object-Oriented model: [11], [27], [24], [17], [1];
- Semantic Data model: [16], [13], [6], [2];
- Relational model: [37], [15], [20], [23], [29].

The SUPER project is based on ERC+, an object-based extension of the entity-relationship model designed to support complex objects and object identity [33]. The goal of this project is to produce an integrated CASE tool (whose underlying model is ERC+) supporting interactions during all the life cycle of a database. To that extent, we first built a graphical definition and manipulation interface, providing users with a consistent approach to both functionalities. A data manipulation and a data browsing tool will deal with updates and navigation at the occurrences level.
Moreover, a view definition tool will allow users to build views over an existing schema. Conversely, a view integration tool will allow to build an integrated schema from a set of user views. This last tool will be the kernel of a future database design tool, covering, as in DDEW, the various phases in this activity. At this purpose, some others tools are planned for schema normalization, restructuring and evolution.
This global approach will provide the user with the same interaction paradigms all along his/her dialogs with the different components of the DBMS, during both design and operation on the database. Moreover, ERC+ modelling provides the user of a database with the same objects as the real world of interest to him/her.
It should be clear that the main concern of SUPER is to design clear, clean, easy, precise and uniform user interaction methods. SUPER may be used as a front-end to a relational or an object-oriented DBMS. It is not our aim to develop it as a self-contained complete DBMS.

The next section discusses the basic principles governing the design of DBMS user interfaces, and shows which choices have been made in SUPER. Section 3 briefly recalls the characteristics of the ERC+ approach, which will be used in the sequel to illustrate the functionalities of the graphical interface. Section 4 presents the main characteristics of the data definition interface (the schema editor), while section 5 gives an overview of how users can define queries on the database. Finally, the conclusion summarizes the main features of the project and presents future and ongoing extensions.

2 Guidelines for SUPER Interfaces

In the last decade, the need for user-friendly interfaces to large systems (including DBMSs), has generated numerous contributions to the topic. The field has grown mature enough to stress the importance of sound principles for the design of good interfaces (see [31], for instance). This section discusses some aspects we felt are of major concern for visual DBMS interfaces.

2.1 Direct Manipulation

The first goal in visual interfaces is to avoid the use of complex command languages. Some graphical conventions should be adopted, to make visible on the screen both the objects being manipulated and the functions used to manipulate them. Users may interact with the system through direct click and point specifications. This paradigm is known as *direct manipulation* [30] and it is a de facto standard for graphical, bit-mapped workstations provided with a multiwindowing system and a mouse.
The advantages of this approach are: first, users may permanently see the information they work on (for instance, the schema diagram). Secondly, users immediately see the impact of their actions through the visual representation. Finally, users can perform physical actions (like selecting and dragging an object) to modify the graphical representation, or activate dedicated functions (through menus, labelled buttons, dialog boxes and so on) to manipulate available application objects. A direct manipulation interface is easy to learn for novice users, easy to remember for occasional users (with knowledge of the domain of use) and rapid enough for expert users.

2.2 Unconstrained User Behavior

A second important goal is to provide users with freedom from having to follow a predefined pattern in their interaction with the system. Whenever actions are not atomic, the users should be allowed to start some action and move to another one without having completed the first one. Moreover, they should not be compelled to perform a set of actions in a predefined sequence.
Some of the existing prototypes do not adhere to this principle. For instance, the schema editor described in [8] forces users to complete a consistent specification of objects at creation time (e.g. entities with at least one attribute). Many graphical query languages (as the one described in [10] and QBD [7]) impose some fixed sequencing of steps, at least for building the query subschema and for its restructuring. Other interfaces do not explicitly state what are the built-in constraints. Only some less demanding interfaces allow users to leave definition of objects incomplete, as in Schemadesign [28] or in SNAP [6]. Pasta-3 supports a high degree of flexibility in user interactions.
SUPER fully supports unconstrained user behavior, both for schema editing and for query formulation. Each tool is responsible for checking user actions, ensuring the desired level of consistency among the actions.

2.3 Well-Suited Graphical Representations

There is no general agreement about what should be a good graphical representation for displaying schema diagrams, queries or data resulting from query evaluation.
Icons seem to be one of the most appealing visual representations. IBS [12] uses icons for representing object types. In SDMS [15] and SICON [14] icons are appropriately placed in a spatial framework, to browse data in an easy way.

Databases with a large number of object types require the users to memorize a large number of iconsand their manipulation can confuse users [31]. Moreover, when coping with many icons, their design is not an easy task. The result may be that an icon is meaningful for its designer, but not for users. Consequently, iconic languages may require as much, or even more, learning time than a textual representation.
Most of the existing prototypes based on semantic models (ER, SDM or IFO) use graphs to represent the conceptual schema of a database. However, they use different formalisms for schema diagrams. For instance, Pasta-3 [19] characterizes the type of nodes by a different character style (plain for entities, bold for relationships), instead of using different graphic symbols. Schemadesign uses the same graphical notation for multivalued attributes as for relationships, which can be rather misleading for users. In ZOO [27] knowledge is represented by a graph of icons, where edges are associated to either classes or objects and arcs represent the relationships between items. Again, users are confronted with the problem of distinguishing items through a large number of different icons.
Some prototypes use colors or patterns to display objects. For instance, GUIDE [35] represents with different colors partial queries embedded in a complex query. DDEW [26] uses different colors and patterns to express links cardinality. ISIS [13] associates to each class an unique fill pattern; this pattern can be used in an attribute definition to express its value class (i.e. a reference), with eventually a white border if the attribute is multivalued. However, the use of patterns is a little clumsy and not immediate for the user. As for icons, the automatic generation of patterns may cause problems when the number of classes becomes too large.
We took simplicity and minimality as guidelines. SUPER keeps the basic ER symbols: rectangles for entity types, diamond boxes for relationship types. Attributes are simply displayed as names attached to the parent object by a line (different line drawings are used according to cardinalities). Generalizations are shown with usual arrows. These symbols are well understood by users.

2.4 Multiple User Profiles

Human-computer interfaces should take into account the existence of several categories of users [31]. Novice or occasional users need basic functionalities accessible through easy to understand graphical displays. For data definition, for instance, these users will build small schemas, picking graphical symbols and putting them together as in a drawing tool. Expert users might favour the definition of a schema via menus and dialog boxes. Moreover, it should be possible to create a schema definition from a textual file imported from some other tool: in this case, the graphical approach is of no use.
To support diverse interaction styles, SUPER provides two modes of operation for schema definition, each mode having its associated window. The graphical mode is based on direct manipulation, while in the alphanumeric mode schema objects are defined through dialog boxes. There is no notion of "mode switch", as both modes are active in parallel: users may freely go from one to the other, any time during the interaction. When the graphical window is used, dialog boxes are prompted with default information, which users may change, if needed. If objects are defined in the alphanumeric mode, a corresponding diagram is automatically generated. Therefore, the two modes are equivalent, the schema editor keeping them synchronized: all visible representations of the same object are automatically updated when users modify its definition. These two representations are different ways to show the overall schema (i.e. they can both be used to display all available information). They are not

complementary representations of different aspects of the schema.
Some of the existing DBMS interfaces do have a notion of mode switch and do not allow for a global view of the schema. For instance, in [2] users have to enter a textual environment to specify properties of classes, whereas classes and their relationships can be specified only in a graphical environment: the two representations are neither simultaneously displayed nor equivalent. In Schemadesign there are different modes for the definition of entities and relationships on the one side, and their properties on the other side. The ER graph and the inheritance lattice used by Pasta-3 are an example of complementary, but not equivalent representations of a diagram. ISIS provides several different views of a schema, as the so-called "inheritance forest view", in which all class definitions are displayed; users may expand an object into the associated semantic network by clicking on it. Again, the main shortcoming is the impossibility of having a simultaneous display of the two representations and even of the semantic networks associated to two different classes.

2.5 Consistent Paradigms

User interfaces should avoid different modes of operation, or different dialog styles, when switching from one function to the other. For instance, it is usual to base data manipulation on the same schema representation as the one used at schema definition time [35] [36] [10] [13] [6] [7] [9].
However, this kind of consistency is not always achieved. For instance, several interfaces use the ER paradigm for data modelling, while offering manipulation facilities which are close to those defined for relational databases, or presenting the resulting data as relational tuples rather than ER objects [10] [21] [7] [9]. Pasta-3 supports query formulation directly on the schema diagram, but also uses QUEL-like displays for predicates [18]. A consistency example may be found in SNAP, where the same formalism ("comparitor arcs") is used both for the conceptual schema and for predicate specification: however, only very simple predicates can be specified.

2.6 Assertional Data Manipulation

The experience from textual interfaces shows that assertional languages are to be preferred, w.r.t. procedural ones, especially for non expert users. Visual DBMS interfaces should therefore depart from requiring queries to be formulated as a strict sequence of operations (programming steps), to be executed in the order they are specified. In [9] queries are specified as a sequence of operators (graphical counterpart of an underlying algebraic language), which transform at each step the subschema into a new one. On the contrary, users should be allowed to independently specify the different components of their query. This not only alleviates the user's task, but also leaves the editor with the possibility of optimizing query processing. Moreover, users should be able any time to modify any stated part of the query, either to correct errors or to refine some incomplete specification. This seems to us the only way to put into practice, for data manipulation, the unconstrained user behavior principle.

2.7 Object Management

The evolution of modelling requirements has highlighted the need of keeping application objects when implemented onto a DBMS. According to the principle of consistent paradigms, data manipulation interactions should support objects, without making them vanish into a set of relations.
While in relational interfaces a query defines a single resulting relation, in object-

based interfaces the resulting object type is an attribute tree showing only value attributes. As reference attributes bear object identities, they have to be replaced in the resulting structure by either an object identifier or the whole value of the referenced object (referenced object are embedded into the referencing one).
In visual database interfaces, the structure of the result is defined by building the so-called *query subschema*, i.e. the desired restriction of the underlying database schema. This subschema can be unambiguously interpreted if it has a hierarchical structure, with no cycles in it. If a cycle is kept, to express recursion, query formulation must explicitly state the path to follow to explore the cycle (linearization of the cycle).
Relational-like visual interfaces can support graphs as final query subschemas, as they produce the result by generating a flat join among all relations in the final graph [9]. Object-based interfaces transform the subschema into a hierarchy by identifying one of the object types in the subschema as the "root" of the query [10]. All other object types in the subschema are accordingly turned into attributes of the root object type. SUPER follows this strategy (with appropriate refinements, as discussed in § 5.2).

3. The ERC+ Model

ERC+ is an object-based extension of the entity-relationship model, specifically designed to support complex objects and object identity. Object types may bear any number of attributes, which may in turn, iteratively, be composed of other attributes. The structure of an object type may thus be regarded as an multiple attribute tree. Attribute complexity and multivaluation express the usual product and set constructs of the object-oriented approach, with the advantage that they may simultaneously apply at the same node. Attributes, entities and relationships may be valued in a multiset (i.e. not excluding duplicates). An object identity is associated to entities and relationships, i.e. different instances may have exactly the same values for their attributes. Two generalization relationships are supported on entities: the classical "is-a" and an additional "may-be-a" relationships [32]. The former corresponds to the well-known generalization concept; the latter has the same semantics, but does not require an inclusion dependency between the subtype and the type. A complete discussion of the features of the model may be found elsewhere [33] [32].

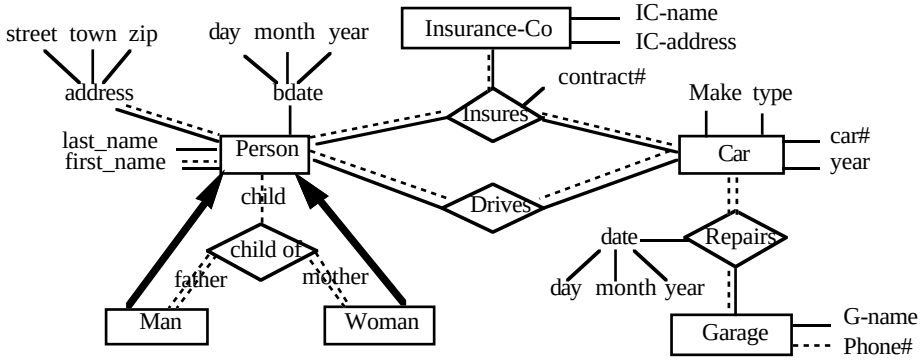

Fig. 1. A sample ERC+ schema

Figure 1 shows a sample ERC+ diagram; a single continuous line is used to represent a 1:1 link, a single dotted line represents a 0:1 link, a double dotted line represents a

0:n link, a double line (once dotted, once continuous) represents a 1:n link. Arrows represent generalizations.
Formal manipulation languages (an algebra and an equivalent calculus [25]) have been defined. The functionalities provided by the algebraic operators include the classical operations of *projection* and *selection* on one entity type and *union* of two entity types.Specific to ERC+ is the *reduction* operator, which allows the elimination of the values of an attribute not satisfying a given predicate. Most important is the relationship-join (*r-join*, in short) operator. If E1, E2, ..., En is the set of entity types linked by a relationship type R, the r-join of E1 with E2, ..., En via R builds a new entity type (and the corresponding population) with the same attributes as E1 plus an additional attribute, named R, whose components are the attributes of R, E2, ..., En. A *spe-join* operator allows the joining of entity types participating into a given generalization.
Every operation results in the creation of a new entity type, with its attributes, relationships and population derived from the operands through specific rules. Operations may thus be combined into expressions of arbitrary complexity.

4 Schema Editor

The schema editor is a visual data definition interface, providing two modes for schema definition. Each mode has a separate display window, identified by the name of the schema being edited and labelled by the corresponding operation mode. Users may work simultaneously on several schemas with different modes. In the *graphical* mode, the designer builds an ERC+ diagram by direct manipulation. The user picks the graphical symbols from a palette and positions them into the workspace provided in the associated window. The symbols in the palette correspond to ERC+ constructs (entity, relationship, link, attribute, generalization). In the *alphanumeric* mode, forms-like representations of ERC+ constructs (called *object boxes*) are provided by the editor for entering data definitions. Different object boxes are shown in figure 2.
Standard editing operations are available through pull-down menus. "Schema", "Edit" and "Dictionaries" menus are available in both windows, and provide the same functionalities. The "Schema" menu contains the usual operations for opening, saving, creating a schema. The "Edit" menu offers cut, copy and paste facilities, as well as undo and redo. The "Dictionaries" menu gives access to a global dictionary or any of the specialized dictionaries (entities, relationships, attributes). The "Options" menu in the graphical window contains purely graphical manipulations (changing the layout, rearrange object disposal, etc.) and is therefore specific to this window. Conversely, functionalities for creating a new schema (or modifying an existing one) are provided in the "Creation" menu when in the alphanumeric mode. They are equivalent to the definition of schema elements through the graphical palette.

4.1 Information Display

Figure 2 shows the schema diagram for a hypothetical application for an Insurance company. Each object in the diagram has been created by first selecting the corresponding graphical symbol in the palette and then clicking in the workspace to position the object. Creating an object displays the corresponding object box (alternatively, it may be displayed using the alphanumeric Creation menu). Newly created objects receive a standard name, which can be changed in the corresponding alphanumeric object box. An object box contains text entry areas (e.g. object's name

and comment), radio buttons for predefined choices (cardinality specification, for instance) and list-bars referring to objects directly attached to the current object. The entity box (Person) in figure 2 shows list-bars for attributes, links and generalizations defined on an entity type. A list-bar for components of a complex attribute is included in the attribute box (address). List-bars have been chosen as a standard technique to link objects. Clicking on a list-bar displays the corresponding scrollable list of attached objects. Two such lists are shown in figure 2, one for links on the Person entity type, one for components of the address attribute.

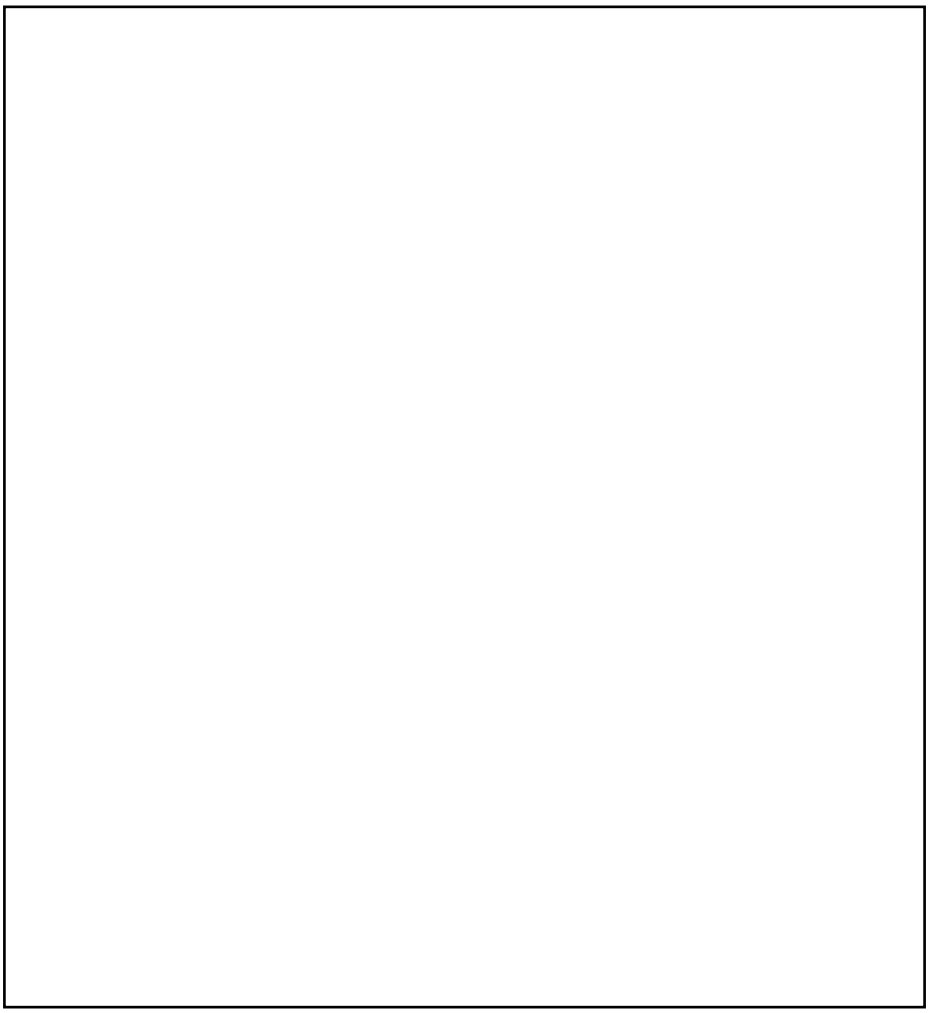

Fig. 2. Screen display showing schema editor windows
(after creation of a new attribute, whose default name is att1)

Lists have a standard behavior. They group objects of the same type, attached to the same parent element (the latter is the schema for dictionary lists). Clicking on an object in a list displays its object box. Clicking on the New button in the list box displays an empty object box for adding a new object to the list. Using object boxes,

list-bars and the attached lists, users may navigate through the schema and add or modify objects as needed. Top-down definition strategies are very easily performed.
ERC+ diagrams are stored with spatial information, so that the diagram may be later displayed as it was at creation time. For schemas defined in the alphanumeric mode, SUPER automatically builds and displays the corresponding diagram. As governing this process is a complex task, our implementation leaves it up to the users to adjust the diagram if they dislike it. For readability, the user may hide attributes.
Flexibility in the schema design process is enhanced by the possibility to leave object definitions incomplete. Users may, for instance, define entity types, and come back later to these objects to attach attributes or add generalizations. Incomplete schema definitions may be saved and reused in another session. At any time, a validation function may be activated to check whether the actual schema definition is consistent with model rules. If inconsistencies or incompleteness are detected, they are reported to the user. However, some model rules have to be permanently enforced (uniqueness of entity names, for instance) in order to avoid ambiguities.
Finally, users may quit the editor any time. When they continue schema editing, their work on the schema is reactivated in exactly the same state as it was at the time of interruption.

4.2 Flexibility, Reusability and Backtracking

Besides the above usual editing functions, SUPER includes the additional facilities of redundancy, reusability and backtracking to make users' task as easy as possible.
Redundancy is intended to provide flexibility. It allows the users to view their schema through two equivalent representations. Accordingly, some functionalities have been implemented redundantly, so that users may access them directly through the representation they are using. For instance, an object may be created graphically, or in several alternative ways in the alphanumeric mode. A new attribute, for instance, can be created either by activating the Creation menu, or clicking on the New button in the attribute list attached to its parent object. Whichever way is used, it will result in displaying an attribute creation box, where users will enter the attribute definition.
This kind of flexibility is sometimes criticized as being confusing for users. We believe it might indeed be confusing if the alternatives appear within a single context, with users not having a criteria to choose from. On the contrary, if the alternatives are provided along different paths, it avoids the burden of explicitly moving from the context they are in to the context which provides the desired function.
Reusability allows the users to reuse definitions of objects in the current schema or in another schema. *Cut*, *copy* and *paste* operations may be used to move an object, delete it, or create a similar object elsewhere. The object here may be a single object (an entity type, ...), or a collection of objects (a set of attributes may be copied from various existing objects and in one shot attached to an entity type), or a subschema (a set of interrelated objects obeying some given model constraints: for instance, no isolated attributes, links and generalizations must be with their source and target objects). A *duplicate* operation is provided. It creates an object identical to the original one and bearing the same connections.
Finally, backtracking is supported through *undo* and *redo* operations. This allows the users to recover from erroneous actions and restore the previous state. Typically, if a user clicks on a Cancel button instead of the nearby OK button, all actions performed on the object would be lost. By undoing the erroneous click, he/she will get a second chance.

4.3 Browsing

The current version of SUPER supports schema browsing. Users may scroll the schema diagram to display the desired part of it. The alphanumeric mode allows schema browsing by navigation from one object to another through existing connections in between. This navigation may use object boxes (as shown in figure 2) to allow user to see all informations about the objects on the path. A similar navigation may also be performed using a simultaneous display of the various dictionary lists. For instance, the selection of an entity type in the entity types list will automatically display its attributes, relationships and generalizations or specializations, if any, in the corresponding list. However, the only information users get in such a navigation are the names of the objects. To know more about a specific object, users have to click on its entry in the appropriate list, to activate its object box (the information contained in the object box is then only available for inspection, to prevent conflict between different actions on the same object).

5 Query Editor

This section discusses the features concerning query formulation in the SUPER environment. For more details about the query editor, see [4]. The steps which compose this process are the following:

- *Selecting the query subschema*: the portion relevant to the query is extracted from the database schema.
- *Creating the query structure*: the subschema is transformed into a hierarchical structure (as discussed in § 2.7).
- *Specifying predicates*: predicates are stated on database occurrences, so that only relevant data is selected.
- *Formatting the output*: the editor is provided with data items to be included into the structure of the result;
- *Displaying resulting data*.

The whole process may be rather complex, and therefore difficult to master for novice users. As these users are the main target of visual interfaces, we believe that visual query languages should take advantage of the above multistep structure. Indeed, clear separation between the steps alleviates users' mental load and improves the chances of correct formulation. The sequence of steps is logically meaningful: for instance, predicates cannot be defined before the query subschema is determined. Users can any time modify any stated part of the query to correct or refine the current formulation.
SUPER implements step separation, by using the following specific windows for the different steps:

- the *database schema* (DBS) window is a read-only window used to display the diagram corresponding to a schema (figure 3);
- the *working schema* (WS) window displays the subschema, extracted by the user, relevant for a query (figure 5);
- the *selection window* (SW) displays the structure of the resulting entity type, and it is used for expressing predicates and the attributes to be kept in the result (figure 6);
- the *result window* (RW) displays the set of resulting occurrences (figure 8).

5.1 Selecting the Query Subschema

This step configures the schema to contain only those objects which are involved in

the query (equivalent to an "open subschema ..." command in textual languages). The DBS window is used to extract the query subschema through a sequence of "point and click" specifications. To speed up this process, the semantics of the clicks can be tailored either as "keep" or as "delete" the designated object. Implicit designation is sometimes used: gql/ER [36], for instance, automatically adds to the query subschema the path in between two selected objects (in case of multiple paths, the "most likely" one is chosen). QBD [7] uses a similar technique, which can be refined with additional constraints (on the length of the path, for instance). It also allows predicates on attribute names to select all entity types with such attributes.

In SUPER, the "point and click" specifications copy objects in the WS window. The user can choose between a traditional *Copy-Paste* mode (objects are copied without relating them to objects already in the WS) and an *Expand* mode, where objects are copied and connected to a start entity type previously selected in the WS window.
Some automatic selection is embedded in SUPER. If the user clicks on a role, the complete relationship type is transferred in the WS. If the relationship is binary, the start entity type changes to the new one. Clicking on a distant object is possible if there is a smallest path to the object. For instance, the user cannot click on a relationship which has two roles leading to the start entity type.

5.2 Restructuring the Query Subschema

Once a query subschema is defined, proper query formulation may start. However, some interfaces introduce an additional step, to transform the subschema into a specific pattern. In [10] the query subschema is transformed into a hierarchical structure. The root of the hierarchy is selected by user. [21] follows the same approach, but the transformation is complemented with a generation of nested forms, visualizing the hierarchical structure. QBD provides a query-like transformation language for schema modification.

In SUPER, graphical data manipulations are based on the underlying ERC+ algebra. The result of a query is a syntactical tree whose root is an entity with constraints expressed as predicates. In [4] we discuss the use of tree representation of queries.
In our editor, the user identifies the root of a query hierarchy. A graph in a tree is transformed by first removing cycles. The removal of cycles could not be an automatic process as there are many interpretations of the cycle itself.
In the SUPER query editor, the user can break cycles by removing some vertices or some nodes of the graph or by disconnecting some links. *Disconnection* means that the designated link is detached from the linked entity type and attached to a (newly automatically created) copy of that entity type.
Another facility, *pruning*, is used to remove objects (attributes, entity types, ...) which are not used in the query, i.e.appearing neither in the format of the result nor in a predicate. There are some additional facilities like the product to create an artificial link between two entities.

5.3 Specifying the Predicates

Once the user has created a correct query structure, the corresponding hierarchy is displayed in the SW as a single entity type with all other informations as attributes. Appropriate modifications, if needed, can be made in the SW; otherwise, the user will proceed with the specification of predicates.
Predicates against complex objects may be rather clumsy. For the simplest ones (comparison of a monovalued attribute with a constant) a graphical counterpart may

easily be defined. A simple specification technique is to click on the attribute, select a comparison operator from a menu, and finally type the value or choose one from a list. For complex predicates (involving several quantifiers, for instance), there might be no simple way to express it graphically. Menus are sometimes used for syntactic editing of predicates ([10] or ISIS [13]). In gql/ER, QBE-like forms are used to specify conditions on the selected nodes. Only a few interfaces use a graphical formalism for expressing predicates (see, for instance, Pasta-3 [18] or SNAP [6]).
In this paper we focus on functionalities, instead of discussing the best graphical solution for predicate specification. A predicate is any logical expression involving attributes of the entity type resulting from the previous step. The predicate implements the selection operator if it is attached to the root or the reduction operator if it is attached to an attribute. The domain of a predicate is the set of quantified variables.
Predicates are expressed by using a *predicate box* associated to the root of the hierarchy. Subpredicates are automatically generated when the user designates the attributes involved in the predicate. By default (as in Pasta-3), the equality operator is used for clauses and the existential quantifier is assumed for multivalued attributes. Every attribute which appears several times in one or several predicates is duplicated as many times in the SW. This is the graphical counterpart of the use of variables in textual languages.
Evaluation of the predicates can be done in any order. Each intermediate step usually builds a potential query that can be interpreted (a syntactic validation) and executed (a semantic validation on an existing database) in the fourth window.

5.4 Formatting the Output

By default, the SW defines the structure of the resulting entity type. However, the users may wish to discard some of the attributes, kept up to now only because of some predicate to be defined on them. The SW has to provide for a "hide" (or, conversely, "show") operation, to define which attributes are to be discarded (or kept in). The hiding (or the showing) of a complex attribute also hides (or shows) all its component attributes.

5.5 Displaying Resulting Data

The last phase is the display of instances representing the result of the query. SUPER displays resulting entities according to their hierarchical structure, into a nested tabular form (NT mode) or into an entities browser form (EB mode). Users can choose between the two kinds of presentation through a switch mode radio button.
Relational interfaces display occurrences in a tabular form. SNAP provides the user with the choice between the tabular format and a NF^2 format, where occurrences are arranged into "buckets". [21] uses the nested forms representation to display the results of data browsing. GUIDE [35] and VGQF [23] allow the user to choose among different formats. In OdeView [1] complex objects can be displayed through a text or picture representation; the user can click on buttons to display all related objects.

5.7 Additional Facilities

Additional functionalities supported by SUPER allow to store queries for later reuse or modification, as well as evaluation of partial queries. The latter is useful for query debugging. Suppose the user is confronted with a result different from what was

expected, without recognizing what is the problem within the formulation. The query can be broken into two or more separate queries by disconnecting some links. Independent evaluation of the subqueries can be performed to identify what changes have to be made. After this refinement, the original, corrected query can easily be rebuilt by unification of the duplicated entity types created by the previous disconnection.

5.8 A Sample Query

This section illustrates the process of query formulation in SUPER. Let us assume the user wants to formulate the following query:

Select name and address of people who insure a 1984 Ford

on the schema of figure 1. The corresponding diagram will be displayed in the DBS window. The corresponding diagram will be displayed in the database window. The user begins by picking the relationship type Insures, that will be copied into the WS window, together with the linked entity types (Person, Insurance-Co and Car) and all their attributes (figure 3).

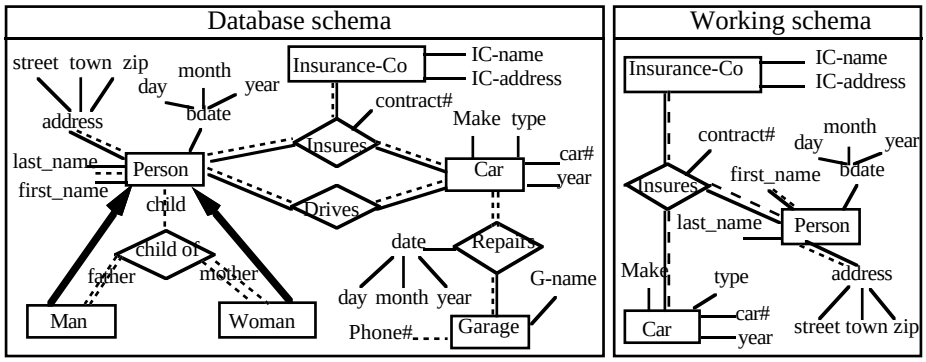

Fig. 3. Query editor windows showing query subschema definition

Next, assume the user designates Person as the root of the hierarchy. Figure 4 shows the contents of the selection window.

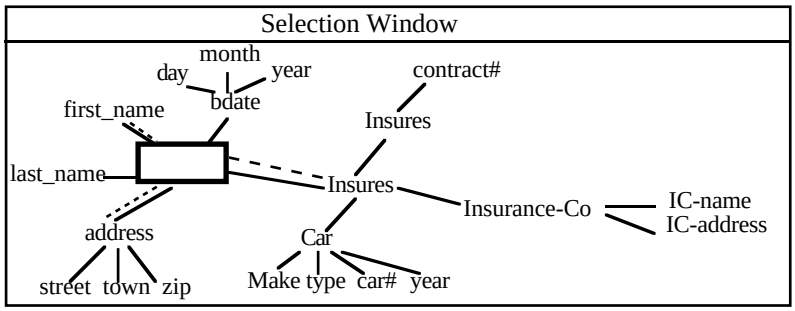

Fig. 4. Resulting hierarchical structure, showing the root as an empty box

The resulting structure contains many attributes the user is not interested in. Consequently, he/she will return to the WS window and prune unnecessary objects.

The attributes name and address of Person are needed for result display, while Make and year of entity Car will be used for predicate specification. Pruning will change the contents of windows, as shown in figure 5.

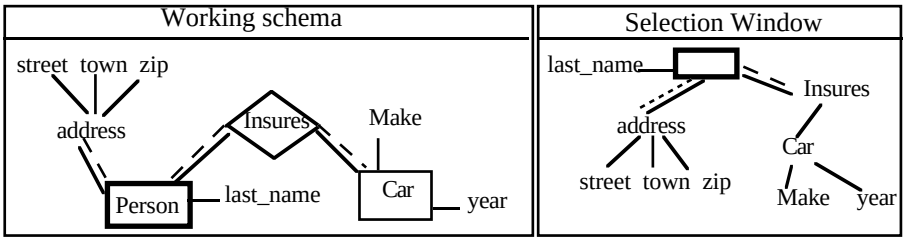

Fig. 5. The updated query subschema and corresponding hierarchical structure

The definition of a predicate is made through a predicate box, displayed in the SW. The user designates the attributes (Make and year) involved in the predicate. As the Insures attribute (to which Make belongs) is multivalued, a modifiable "exist Insures" clause is automatically generated (figure 6).

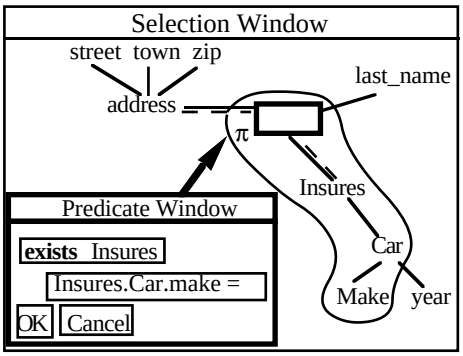

Fig. 6. SW after designation of Make

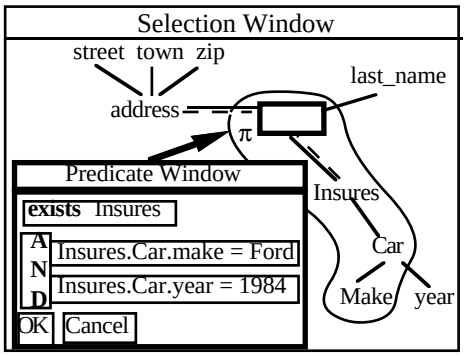

Fig. 7. Final state of the SW

While designating the second attribute (year) the user also has to specify the logical connector between the two predicates. The specification is completed by entering the appropriate values: Ford for Make and 1984 for year (figure 7).The query is ready for evaluation (figure 8).

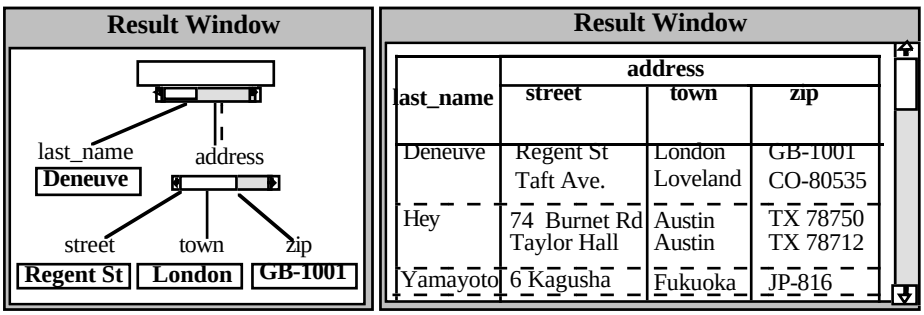

Fig. 8. Result of the evaluated query in the entities browser and the nested tabular form

6 Conclusion

Development of good and powerful visual interfaces is a major current challenge for database system designers. Despite an important investment of research efforts in the general domain of user interfaces, database users are still confronted with textual languages for their interaction with a DBMS. Most of the tools available on the marketplace are limited to data definition facilities. For graphical data manipulation, a few prototypes have been developed to support modern, object-based modelling approaches. However, they do not seem to consistently support all of the desired functionalities. For these reasons we believe more research and experiments are needed to fulfill the goals attached to visual interfaces.

The development of the SUPER environment is intended as a contribution in this direction. Our aim is to use well specified paradigms, consistently over all the phases related to database operation. This includes, of course, data definition and data manipulation, but also considers database design activities, with a particular emphasis on view definition and view integration.

Visual interaction in SUPER environment is based on direct manipulation of objects and functions, providing users with maximum flexibility during schema definition as well as query formulation. Graphical interactions are easy to manage, and take advantage of the support of a simple but powerful modelling paradigm. Visual data manipulation is assertional and object-based. The environment offers multiple interaction styles, well-suited for various categories of users.

The schema editor has been implemented in C++, in a UNIX environment (Sun workstations), complemented with the user interface InterViews [22]. The SUPER environment implementation is based on a toolbox approach. This approach is discussed in [3]. The query editor is currently being implemented. For more details about the architecture of the SUPER environment, see [5].

The next step concerns the specification of a view definition graphical facility. Its first goal will be to allow users to define views over an existing schema. Secondly, we intend to develop a similar tool for definition of views in the initial phase of database design, as a formal way to express user requirements. These views will be used as input to an integration tool, which will automatically perform their integration according to explicitly defined interview correspondences [34]. Finally, more tools will be specified to cover the other phases of the database design process.

Acknowledgments

The authors are indebted to Prof. Bharat Bhargava and Prof. Kokou Yétongnon for many useful suggestions to previous versions of this paper. They also want to express their gratitude to Claude Vanneste and Bull France for their cooperation in the ergonomics of SUPER editors.

References

[1] R. Agrawal, N. H. Gehani, J. Srinivasan: "OdeView: The Graphical Interface to Ode", in *Proc. of ACM SIGMOD '90, Int'l Conf. on Management of Data*, pp. 34-43, Atlantic City, 1990

[2] A. Albano, L. Alfò, S. Coluccini, R. Orsini: "An Overview of Sidereus, a Graphical Database Schema Editor for Galileo", in *Advances in Database*

Technology - EDBT '88, J. W. Schmidt, S. Ceri, M. Missikof eds., Springer-Verlag, pp. 567-571, 1988

[3] A. Auddino, E. Amiel, B. Bhargava: "Experiences with SUPER, a Database Visual Environment", in *Proc. of the 2nd Int'l Conf. on Database and Expert Systems Applications - DEXA '91*, pp. 172-178, Berlin, 1991

[4] A. Auddino, Y. Dennebouy, Y. Dupont, E. Fontana, S. Spaccapietra, Z. Tari: "SUPER: A Comprehensive Approach to Database Visual Interfaces", in *Proc. of IFIP WG 2.6 2nd Working Conf. on Visual Database Interfaces*, pp. 359-374, Budapest, 1991

[5] A. Auddino, E. Amiel, Y. Dennebouy, Y. Dupont, E. Fontana, S. Spaccapietra, Z. Tari: "Database Visual Environments based on Advanced Data Models", in *Proc. of the Int'l Workshop on Visual Interfaces - AVI '92*, pp.., Rome, 1992

[6] D. Bryce, R. Hull: "SNAP, a Graphics-Based Schema Manager", in *Proc. of the 2nd IEEE Int'l Conf. on Data Engineering*, pp. 151-164, Los Angeles, 1986

[7] T. Catarci, G. Santucci: "Query by Diagram: A Graphic Query System", in *Proceedings of the 7th Int'l Conf. on Entity-Relationship Approach*, pp. 157-174, Rome, 1988

[8] E. Chan, F. Lochovsky: "A Graphical Database Design Aid Using the Entity-Relationship Model", in *Entity-Relationship Approach to Systems Analysis and Design*, North-Holland, pp. 259-310, 1980

[9] B. Czejdo, R. Elmasri, D. W. Embley, M. Rusinkiewicz: "A Graphical Data Manipulation Language for an Extended Entity-Relationship Model", *IEEE Computer*, Vol. 23, No. 3, pp. 26-36, March 1990

[10] R. A. Elmasri, J. A. Larson: "A Graphical Query Facility for ER Databases", in *Entity-Relationship Approach - The Use of ER Concept in Knowledge Representation*, P. P. Chen ed., North-Holland, 1985, pp. 236-245

[11] D. H. Fishman et al.: "Iris: An Object-Oriented Database System", *ACM Transactions on Office Information Systems*, Vol. 5, No. 1, pp. 48-69, January 1987

[12] C. Frasson, M. Er-radi: "Principles of an Icons-Based Command Language", in *Proc. of ACM SIGMOD '86, Int'l Conf. on Management of Data*, pp. 147-151, Washington, 1986

[13] K. J. Goldman, S. A. Goldman, P. C. Kanellakis, S. B. Zdonik: "ISIS, Interface for a Semantic Information System", in *Proc. of ACM SIGMOD '85, Int'l Conf. on Management of Data*, pp. 328-342, Austin, 1985

[14] I. P. Groette, E. G. Nilsson: "SICON, an Iconic Presentation Module for an E-R Database", in *Proc. of the 7th Int'l Conf. on Entity-Relationship Approach*, pp. 137-155, Rome, 1988

[15] C. F. Herot: "Spatial Management of Data", *ACM Transactions on Database Systems*, Vol. 5, No. 4, pp. 493-514, December 1980

[16] R. King, S. Melville: "SKI: A Semantics-Knowledgeable Interface", in *Proc. of the 10th Int'l Conf. on Very Large Databases*, pp. 30-33, Singapore, 1984

[17] R. King, M. Novak: "FaceKit: A Database Interface Design Toolkit", in *Proc. of the 15th Int'l Conf. on Very Large Data Bases*, pp. 115-123, Amsterdam, 1989

[18] M. Kuntz, R. Melchert: "Pasta-3's Graphical Query Language: Direct Manipulation, Cooperative Queries, Full Expressive Power", in *Proc. of the 15th Int'l Conf. on Very Large Data Bases*, pp. 97-105, Amsterdam, 1989

[19] M. Kuntz, R. Melchert: "Ergonomic Schema Design and Browsing with More Semantics in the Pasta-3 Interface for E-R DBMSs", in *Entity-Relationship*

Approach to Database Design and Querying, F. Lochovsky ed., North-Holland, 1990

[20] J. Larson, J. B. Wallick: "An Interface for Novice and Infrequent Database Management System Users", in *AFIPS Conference Proceedings, National Computer Conference*, vol. 53, pp. 523-529, 1984

[21] J. Larson: "A Visual Approach to Browsing in a Database Environment", *IEEE Computer*, vol. 19, no. 6, pp. 62-71, June 1986

[22] M. A. Linton, J. M. Vlissides, P. R. Calder: "Composing User Interfaces with InterViews", *IEEE Computer*, vol. 22, no. 2, pp. 8-22, February 1989

[23] N. H. McDonald: "A MultiMedia Approach to the User Interface", in *Human Factors and Interactive Computer Systems*, Y. Vissiliou ed., Ablex Publishing Corp., 1984, pp. 105-116

[24] A. Motro, A. D'Atri, L. Tarantino: "The Design of KIVIEW: An Object-Oriented Browser", in *Proc. of the 2nd Int'l Conf. on Expert Database Systems*, pp. 17-31, Tysons Corner, 1988

[25] C. Parent, H. Rolin, K. Yétongnon, S. Spaccapietra: "An ER Calculus for the Entity-Relationship Complex Model", in *Entity-Relationship Approach to Database Design and Querying*, F. Lochovsky ed., North-Holland, 1990

[26] D. Reiner et al.: "A Database Designer's Workbench", in *Entity-Relationship Approach*, S. Spaccapietra ed., North-Holland, 1987, pp. 347-360

[27] W.-F. Riekert: "The ZOO Metasystem: A Direct Manipulation Interface to Object-Oriented Knowledge Bases", in *Proc. of European Conf. on Object-Oriented Programming - ECOOP '87*, pp. 145-153, Paris, 1987

[28] T. R. Rogers, R. G. G. Cattell: "Entity-Relationship Database User Interfaces", in *Proc. of the 6th Int'l Conf. on Entity-Relationship Approach*, pp. 323-335, New York, 1987

[29] L. A. Rowe, P. Danzig, W. Choi: "A Visual Shell Interface to a Database", *Software - Practice and Experience*, vol. 19, no. 6, pp. 515-528, June 1989

[30] B. Shneiderman: "Direct Manipulation: A Step Beyond Programming Languages", *IEEE Computer*, vol. 16, no. 8, pp. 57-69, August 1983

[31] B. Shneiderman, *Designing the User Interface - Strategies for Effective Human-Computer Interaction*, Addison-Wesley, 1987

[32] S. Spaccapietra, C. Parent, K. Yétongnon, M. S. Abaidi: "Generalizations: A Formal and Flexible Approach", in *Management of Data*, N. Prakash ed., Tata McGraw-Hill, 1989, pp. 100-117

[33] S. Spaccapietra, C. Parent: "ERC+: an Object-based Entity-relationship Approach", in *Conceptual Modelling, Databases and CASE: an Integrated View of Information Systems Development*, P. Loucopoulos, R. Zicari eds., John Wiley, 1992

[34] S. Spaccapietra, C. Parent: "View Integration: A Step Forward in Solving Structural Conflicts", *IEEE Transactions on Data and Knowledge Engineering*, 1992

[35] H. K. T. Wong, I. Kuo: "GUIDE: Graphic User Interface for Database Exploration", in *Proc. of the 8th Int'l Conf. on Very Large Databases*, pp. 22-32, Mexico City, 1982

[36] Z. Q. Zhang, A. O. Mendelzon: "A Graphical Query Language fort Entity-Relationship Databases", in *Entity-Relationship Approach to Software Engineering*, Davis et al. eds., North-Holland, 1983, pp. 441-448

[37] M. M. Zloof: "Query By Example", in *AFIPS Conference Proceedings, National Computer Conference*, vol. 44, pp. 431-438, 1975