1989

A GRAPHICAL QUERY LANGUAGE FOR SEMANTIC DATA MODELS

M. Schneider
Universita de Clermont-Ferrand II

C. Trepied
Universita de Clermont-Ferrand II

A GRAPHICAL QUERY LANGUAGE FOR SEMANTIC DATA MODELS

M. Schneider
C. Trépied
Laboratoire Informatique
Université de Clermont-Ferrand II

ABSTRACT

This paper presents a graphical query language for semantic data models. This language is based on a generic semantic model. It is integrated in the CANDID system, which is a graphical interface for the manipulation of databases. Any schema is represented by a graph divided into families of object types. The graphical query language makes it possible to express a request by describing with a graph an object solution of the request. A request is sometimes expressed in several stages. A stage corresponds to the expression of a simpler subrequest. This language is also characterized by the absence of boolean operators and quantifiers, which are replaced by manipulations on Venn diagrams. Since a request in fact defines derived elements, almost all of a request can be incorporated into the schema. CANDID has been designed to be adapted to any DBMS which has deduction mechanisms.

1. INTRODUCTION

As information and communication functions grow, suitable man-computer interfaces need to be made available to final users. For some time already, progress has been made towards offering easy-access interfaces to a database for non-specialist users. Interfaces based on interactive graphical manipulations form an important class because of the obvious properties of user friendliness they can offer.

First, interfaces such as FORAL-LP, QBE (Zloof 1977), LAGRIF (Miranda and Nsonde 1982) based on the relational model were designed. Nevertheless major difficulties remain for the user as shown by human engineering studies (Corson 1983). These concern mainly the expression of joins and the manipulation of boolean operators and quantifiers. The Universal Relation model led to the definition of better adapted languages -- PICASSO (Kim, Korth and Silberschatz 1988) for example -- but still suffers from a lack of semantic expressivity of the underlying data model.

The combined development of semantic models (Hull and King 1987; Potter and Trueblood 1988) and object oriented DBMS offers a new framework. It led to the emergence of a new generation of graphical interfaces. G-WHIZ (Heiler and Rosenthal 1985), ISIS (Goldman et al. 1985), and SNAP (Bryce and Hull 1986) are the most representative systems of this generation, which includes CANDID.

The semantic model of CANDID makes it possible to define schemata which offer the user a natural view on the part of the world which is modeled. A schema is represented graphically by means of a set of symbols, which are limited for the sake of clarity and legibility. To permit an automatic drawing, the schema is not represented by a connected graph, but is divided into families of object types. A family is defined as a set of types linked by ISA relationships. The graph of a schema is accompanied by a textual description which forms a very valuable document. At any time the user has selective access to it.

The graphical interface of CANDID allows for search and update. Two extreme types of final users can be distinguished. First, there are those who occasionally need to search for information. Their need is often unpredictable and urgent. Second, there are those who are more demanding and who want to assure direct responsibility on their information. They consult the base more frequently, often formulate the same requests and want to be able to update their data without the systematic services of a specialist. CANDID is designed to satisfy these two types of users.

In this article, we present only the inquiry features of the interface. The graphical inquiry language is based on the following principles:

- replacement of boolean operators and quantifiers by manipulations on Venn diagrams;
- initiative coming preferably from the system;
- immediate graphical effect of any command;
- possibility of an incremental request composition (re-use of previous stages).

Generally a request is expressed by describing through a graph an object solution of the request. Sometimes it is

made up of several stages, each stage in fact corresponding to a simpler subrequest.

The article is organized as follows. Section 2 is a brief reminder of the semantic model of CANDID. Section 3 presents the screen composition offered by the interface, Section 4 presents the inquiry model, and Section 5 presents the main commands of CANDID. A few examples of requests are given in Section 6. The conclusion briefly compares CANDID with other graphical systems based on semantic models.

2. OVERVIEW OF THE SEMANTIC MODEL OF CANDID

The semantic model of the CANDID system is a generic model for which a detailed description can be found in Schneider and Trépied (1989). Here we shall simply illustrate its different concepts by means of the following example: the organization of a car rally (the "Paris-Dakar" rally, for example). Different kinds of VEHICLEs can be used: motorbike, car, truck. The database must allow preservation of the participants for each year of the rally (the RALLYMAN) and following the participants of the current rally (the COMPETITORs). A TEAM is made up of one or several competitors, the number of competitors depending on the type of vehicle (for example, one for a motorbike, two for a car, three for a truck). A competitor must be recommended by a participant from one of the previous rallies. Finally, any PERSON directly or indirectly concerned with the rally (organizer, follower, person to contact in an emergency), must be also to be stored in the database. The conceptual schema of this database is presented in Figure 1. Figure 2 gives an external schema called REGISTERED. It should be noted that this schema is decomposed in different parts, each part corresponding to a family.

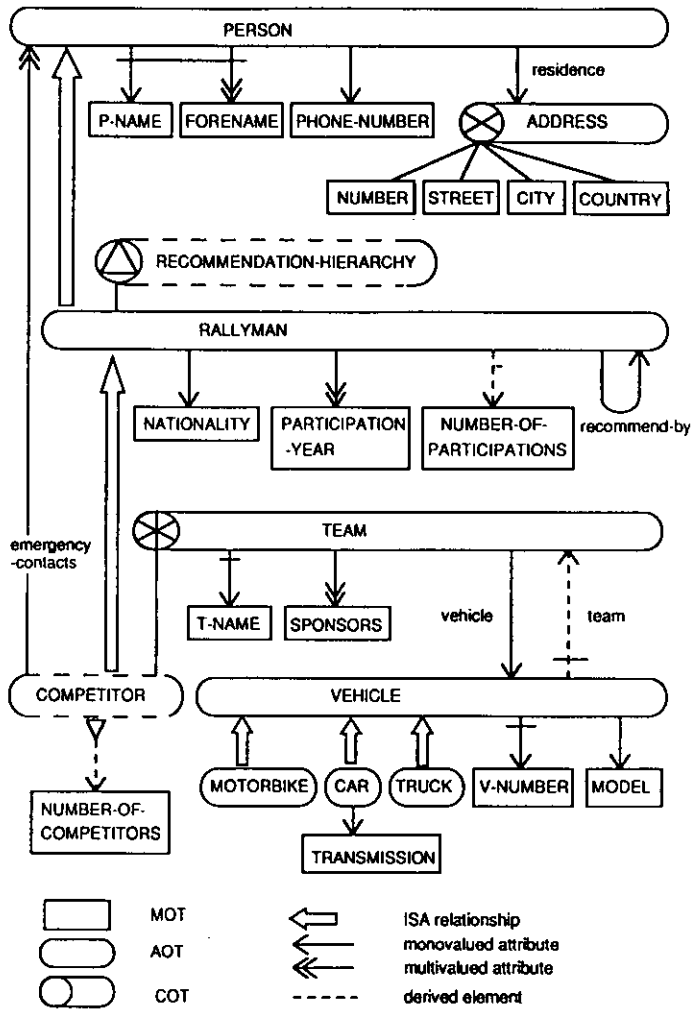


Figure 1. The Conceptual Schema of the Database "RALLY"

The model distinguishes three kinds of types: the mediatic object type (MOT), the abstract object type (AOT) and the constructed object type (COT).

A MOT is used for input into and output from the database. It is the only type for which there is perfect equivalence between the object and its representation through a medium. Only an object of this type can thus be entered or delivered directly through a medium. Several basic MOTs are predefined: INTEGER, REAL, BOOLEAN, TEXT, IMAGE, SOUND. A MOT is represented by a rectangle. For example, P-NAME is a MOT for which every instance is a person's name.

An object of the type AOT cannot be printed or displayed on a screen but only described. In the real world which we want to model, an AO can exist in concrete form (material object) or in abstract form (idea, event). An AOT is described by its attributes. It is represented by a flattened ellipse. For example, PERSON is an AOT.

An object of the type COT is constructed from other objects in the base. The semantics of this assembly are clear, being natural in the real world. Several construction relationships are defined: aggregation, grouping, hierarching. A COT is represented by a flattened ellipse containing the symbol of the constructor type used. For example, ADDRESS is an aggregation where each object is a quadruplet defined on number, street, city, and country; TEAM is a grouping where each object is a set of competitors; RECOMMENDATION-HIERARCHY is a hierarching where each object is a tree of rallymen. In this last case, the root of the tree is a participant in the first rally. The succession relationship used to defined the tree is the inverse relationship of "recommend-by."

An attribute is used to describe one aspect of an object type (the source type) by a semantic link with another object type (the target type). It is represented by an arrow from the source type to the target type. The target type may be identical to the source type. For example, each

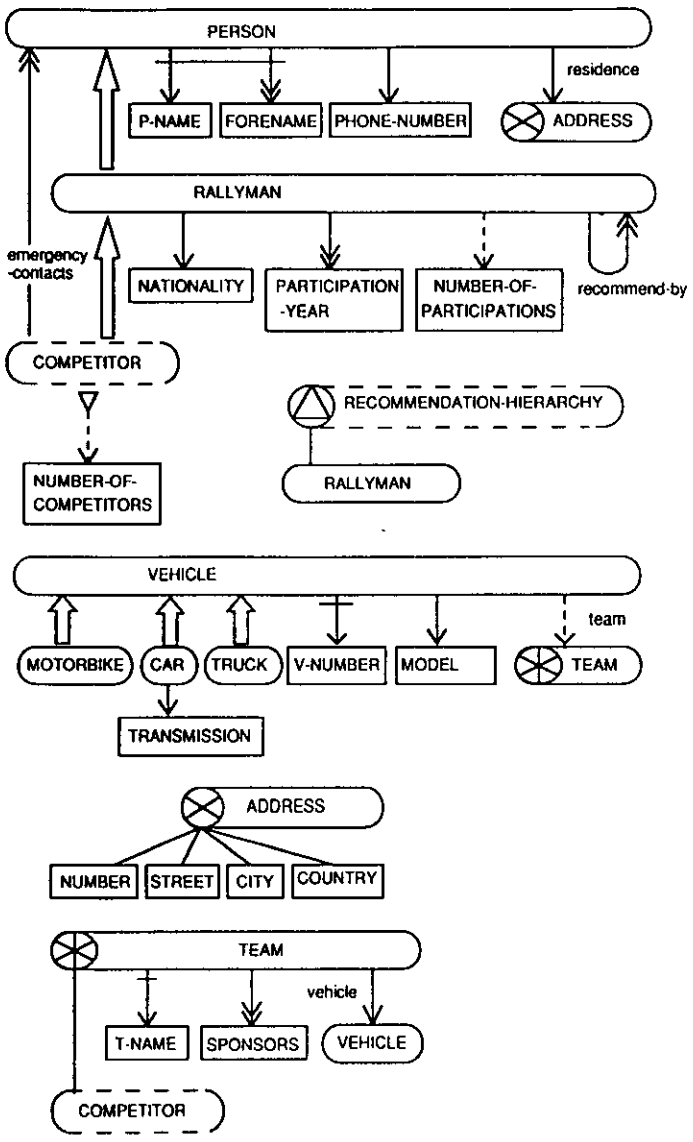


Figure 2. External Schema "REGISTERED" in the Schema Window

rallyman is "recommended-by" another rallyman. An attribute can have an explicit name. Otherwise the attribute name is the same as that of the target type. Two types of attribute can be distinguished:

- object attribute, which describes each object of the source type ("residence" of PERSON, for example)
- class attribute, which describes the class of the source type, the class being the set of objects of this type ("number-of-competitors" of COMPETITOR, for example).

An attribute can be viewed as an oriented relation between the source and the target. It is always possible to define the inverse attribute (i.e., the inverse relationship). CANDID defines automatically the inverse of each attribute. The name of the inverse is implicitly stated to *inv(direct-name)* if *direct-name* is the name of the direct attribute.

An attribute possesses several facets which describe it completely. Those appearing on the graph are

- the name
- the target type
- the cardinality

A monovalued attribute sets up a relationship between a source type object and a single target type object. A multivalued attribute sets up a relationship between a source type object and several target type objects.

Other facets can be defined on an attribute (Schneider and Trépied 1989). Several specify structural integrity constraints. Facets of the inverse are not independent of those of the direct attribute.

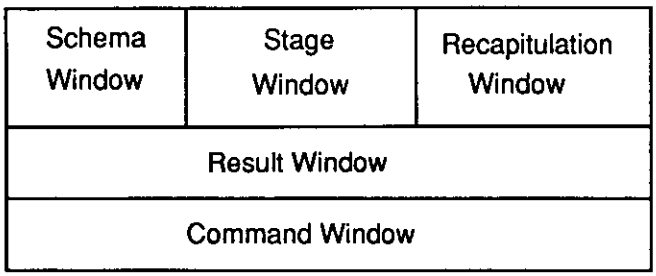
Any object possesses an identity and an aspect. Identity expresses the existence of the object. It is unique and invariable. It is controlled internally by the system. Aspect is expressed through the set of all attributes of the type. An object can have a double (another object with the same aspect). It is possible to forbid doubles. An object can have one or several external identifiers. An external identifier is formed on a subset of attributes. For example, the name and the set of the associated forenames make it possible to identify a person from the RALLY database. An external identifier is represented graphically by a linear junction.

An ISA relationship between two object types indicates that any subtype object "is a" supertype object. For example, RALLYMAN is a subtype of PERSON. A subtype inherits all the object attributes of the supertype. With an ISA hierarchy (for example that of VEHICLE), the static constraints of covering (any supertype object is the occurrence of at least one of the subtypes) and of disjunction (a subtype object cannot be the occurrence of more than one subtype) can be specified. These constraints are defined graphically in CANDID using Venn diagrams.

An element is said to be derived if it is obtained from elements which already exist in the schema. A derivation rule is given to it. Our model allows for derived object types (subtype and COT) and derived attributes. The symbol for a derived element is drawn with dotted lines. For example, COMPETITOR is a derived subtype: a competitor is a participant for whom the set of PARTICIPATION-YEARs includes the current year.

3. SCREEN COMPOSITION

In the inquiry phase, the terminal screen has five windows set out as follows



At any time the user can change the size of one of the first four windows or scroll its contents in any direction.

3.1 Schema Window

The Schema Window contains the graph of an external schema chosen by the user. Starting from the conceptual schema of the database (e.g., the RALLY schema), several external schemata can be defined (e.g., the REGISTERED schema shown in Figure 1). An external schema is constructed from a conceptual schema or a subschema. It can also contain the derived components created by the user while making inquiries on the base. In general, a request for inquiry leads to the definition of several derived elements which are visualized by the system in the Recapitulation Window. The user can memorize definitively these elements by including them in his/her external schema. Each user can thus possess a personalized version of the REGISTERED schema.

The concept of external schema is linked to the concept of semantic relativism (Hammer and McLeod 1981; Tanaka, Yoshikawa and Ishihara 1988): information can be viewed in different ways by one or several people. Logical redundancy is one of the characteristics of our model that facilitates this concept.

In the CANDID system, deduction is carried out by derivation (derived objects are deduced at each request) and not by generation (memorization of derived objects). This choice makes the management of the different external schemata easier.

3.2 Stage Window

A stage is the formulation of a simple request. A complex request is thus expressed in several stages. This means that it is broken down into simple subrequests.

To express a request, the user has a set of commands, all with a graphical effect in the Stage Window, at his/her disposal. He/she constructs progressively the graph of the request. The graph first describes the objects for search in the base and then the information required (objects to be displayed) on these objects.

3.3 Recapitulation Window

Any request for inquiry is in fact a reference to information derived from the database. The specification of a request is therefore equivalent to the specification of derived elements of the schema (types or attributes), without the user being necessarily aware of this. To help the user in his/her approach, the interface offers a recapitulation in a summary form for all the different stages that were necessary to formulate the request.

The Recapitulation Window contains

- The simplified graphs of the derived elements as defined during the different stages.
- The definition of these elements in quasi-natural language (paraphrase proposed by the system). These elements are named by the system. Those defined during the previous stage are marked by an appropriate subscripting.
- The request paraphrased into quasi-natural language.

3.4 Result Window

This window contains the result of the execution of the global request as specified in the Recapitulation Window. There are several possible ways of setting out the results of the request. When several editing formats are detected by the system, they are proposed to the user for a choice.

Information is edited using buckets, for which the semantic expressiveness has been demonstrated (EVER interface [Pauthe 1985] and the SNAP system [Bryce and Hull 1986]). A bucket contains the set of values of a multi-valued attribute. Buckets can overlap so that they can offer the user a natural "non-flat" view of the data (complex objects view).

3.5 Command Window

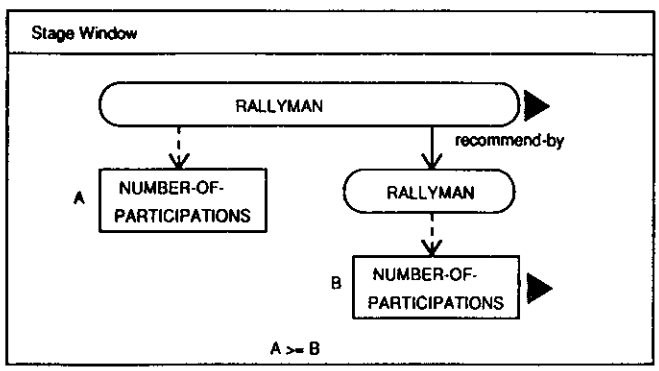
Many of commands are necessary in this type of interface. To facilitate manipulation, the commands are classified into categories. The general menu for these categories is displayed in the Command Window.

By selecting one category, the user obtains a submenu of commands. There is a graphic differentiation between the commands useable at that moment and those which are not.

4. THE INQUIRY MODEL

The basic principle of inquiry in CANDID is that a request is expressed by describing an object solution of the request. The request must first specify an object for search. It must also state which information is to be displayed relative to this object: these are the objects for display.

Example 1: "Which rallymen have participated as often as the person recommending them? State the number of participations of the person recommending them." The request can be formulated in the following way:



An object for search is a rallyman who has participated as often as the person recommending him. Its type is a subtype of the RALLYMAN type. The objects for display (identified by a small triangle) are its identifier (name and forenames) and the number of participations of the person recommending him.

The object for search is specified by constructing a request graph (Banerjee, Kim and Kim 1988) whose elements are similar as those of a schema graph. This graph comprises one or several components. A component is a tree where each node corresponds to an object type and each edge to a relationship (attribute relationship, construction relationship or ISA relationship). The root of the tree describes a unique object type. The tree is obtained from an initial object type (the root node) by applying successively the DEVELOP command. At least one component is used to fix the field of the object for search: the field is the class of the root type. This field can be restricted by some conditions. The conditions are expressed through comparisons. Components are also used to construct the objects or the sets of objects which appear as constant operands in comparisons.

When the graph contains several components, they must be connected with comparisons by using the COMPARISON command.

A request is generally specified through several stages, since in a stage we can handle a maximum of two components. The request is therefore split into subrequests, each of which can be expressed in a stage. There are several reasons for this limitation.

- A complex request is easier to formulate if it is split into simpler subrequests.
- The very nature of the language, which for example does not contain boolean operators, implies the need to split requests.
- The request graph in the Stage Window remains legible and clear.

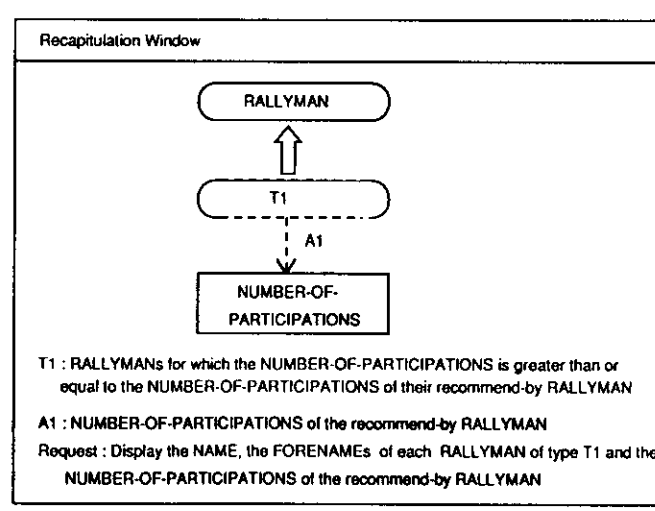
Sometimes the type of the object for search does not exist in the schema: it is therefore a virtual COT, constructed on types from the schema, or a virtual supertype, generalizing types from the schema.

In the case of virtual COT, explicit type derivation commands (GROUPING, HIERARCHING and AGGREGATION) can be used to formulate the request. It should be noted, however, that the mechanism presented above generally allows the user to specify an aggregation implicitly. Thus a graph with two components not connected by a comparison describes a couple of objects, i.e., an aggregation between these objects.

In the case of virtual supertype, since the present version of the interface does not incorporate the definition of supertype, the request is formulated separately on each of the subtypes.

To resume, a stage corresponds to the formulation of a simple request (simple in the CANDID sense). A request is sometimes expressed in several stages. This splitting up is natural as a result of the graphical language. Inquiry with CANDID can be compared to a dialogue where only a limited number of things can be said at once for the sake of clarity. The user expresses his need in the Stage Window, and the system replies in the Recapitulation Window.

Example 2: After the stage presented in example 1, the Recapitulation Window contents are



All of the elements specified during a stage can be re-used in the following stages. The Recapitulation Window contents are used as an extension of the external schema present in the Schema Window.

At the end of each stage, the user can view the result obtained with the request paraphrased in the Recapitulation Window (this result only exists if objects for display have been specified). The request can then be completed by extra stages.

5. THE MAIN COMMANDS

The main commands are classified in five categories, which we shall present in turn.

5.1 Schema Window Commands

LOAD: the system proposes the list of the external schemata to which the user has access. The Schema Window then displays the graph of the schema chosen by the user.

SAVE: the system stores the external schema which has been personalized by the user (including derived elements occurring in the Recapitulation Window).

5.2 Stage Window Commands

These commands are those of the inquiry graphical language.

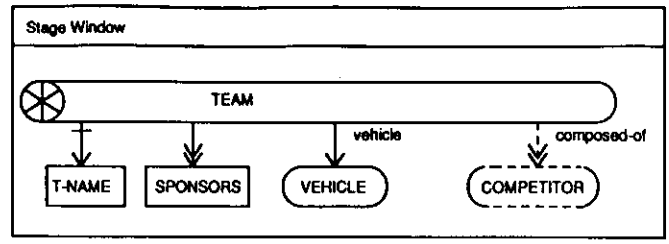
5.2.1 The DEVELOP Command

This command displays the graph of a type selected by the user. The graph of a type T comprises

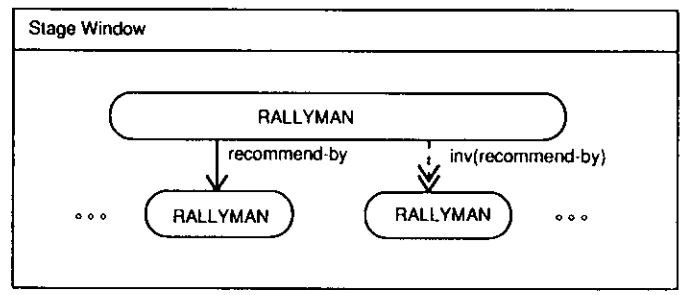
- inherited attributes if T is a subtype
- specific attributes (object attributes and class attributes)
- attributes deduced by the system (implicit inverse attributes, attributes deduced from a construction relationship)

Two cases should be distinguished according to the place (Schema Window or Stage Window) where the user has clicked the type T. If T is clicked in the Schema Window a new component is inserted into the request graph with type T as the root.

Example 3: Contents of the Stage Window after clicking TEAM in the Schema Window



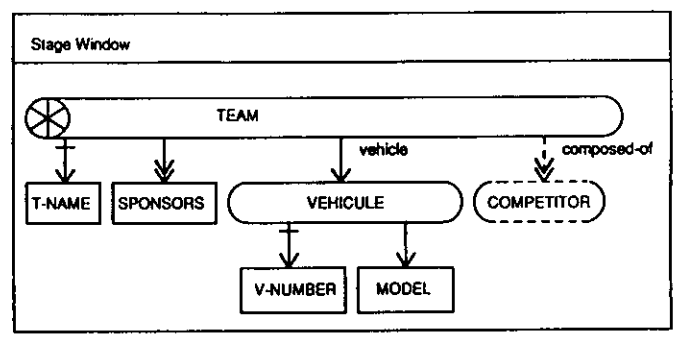
Example 4: Partial contents of the Stage Window after clicking RALLYMAN in the Schema Window



Cyclical attributes are represented in a linear way. In Example 4, the recommended RALLYMAN is clearly differentiated from the recommending RALLYMAN.

If T is clicked in the Stage Window, the command develops the actual graph by replacing the selected node by a complete graph of the type.

Example 5: Contents of the Stage Window after clicking VEHICULE in the graph of Example 3



In this second case, the graph of the type does not contain

- any class attribute if there is no multivalued path (see definition below) in the component from the root node to the T node (e.g., the target attribute "number-of-vehicles")
- any inverse attribute of an attribute already present in the graph.

5.2.2 Selecting a Node in a Component

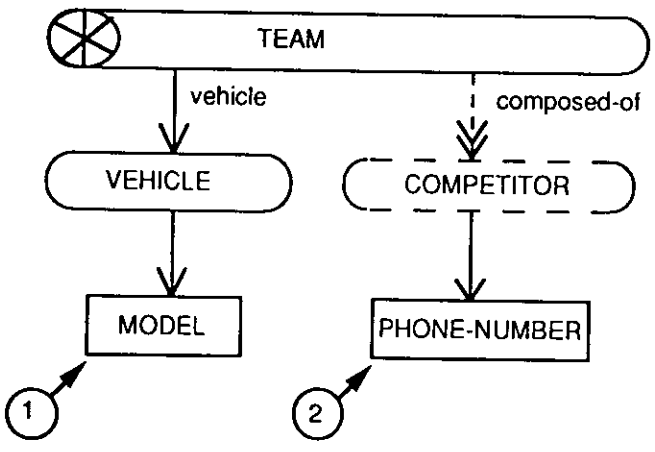
Selecting a node on the request graph in the Stage Window has a specific semantics which is coherent with that of its component: It designates the object or the set of objects of the type selected, attached to a root type object through the semantic links of the path between the root and the node. This path comprises all of the attributes encountered between the root type and the selected type. If at least one of these attributes is multivalued, the selection designates a set of objects (multivalued path). Otherwise the selection designates a unique object (monovalued path). Selecting the root of a component (zero path) designates an object of the root type.

In order to analyze in a correct way the properties of the tree associated to a component, it is important to note the following facts.

- When two types are connected in the schema by a construction relationship, the system manages two attributes inverse of each other: one, called "composed-of," gives the object(s) of the underlying type used by the construction for each CO; the other, called "composes," gives the object(s) of the COT using it for each object of the underlying type.
- When two types are connected in the schema by an ISA relationship, the system manages two attributes inverse of each other: one, called "is-supertype-of," gives, for each object of the supertype, the eventual object of the subtype for which it is the supertype.
- A cyclical attribute in the schema is transformed in a linear way in a component.
- Any attribute has an inverse.

So it is clear that, whichever way a component has been elaborated, the different paths of its tree can be analyzed with the semantics of attributes.

Example 6:



Selection 1 designates an object: the vehicle model of a team.

Selection 2 designates a set of objects: the set of telephone numbers of the members of a team.

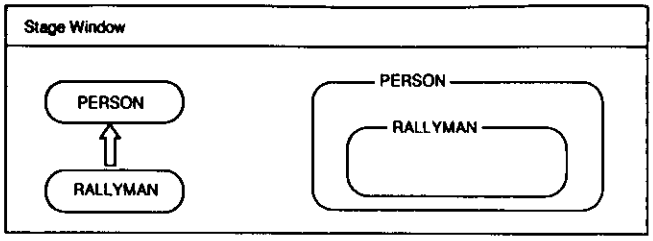
5.2.3 The STIPULATE Command

When a type T is selected by the user there is display of the ISA hierarchy for which T is the root and a combination of Venn diagrams representing type T and its subtypes, respecting the covering and disjunction constraints expressed for this hierarchy. By convention, any zone of the diagram corresponding to an empty subset is shaded grey.

Three cases should be distinguished, according to the place where the user has made his/her selection.

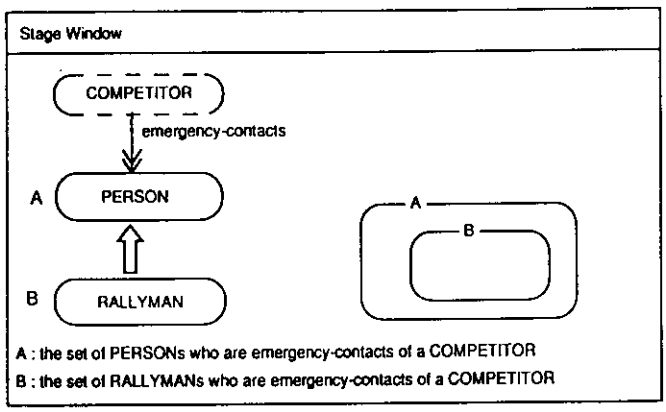
1. If T is selected in the Schema Window or in the Recapitulation Window, then the Venn diagrams represent classes of type T and of its subtypes.

Example 7: Contents of the Stage Window after selecting PERSON in the Schema Window



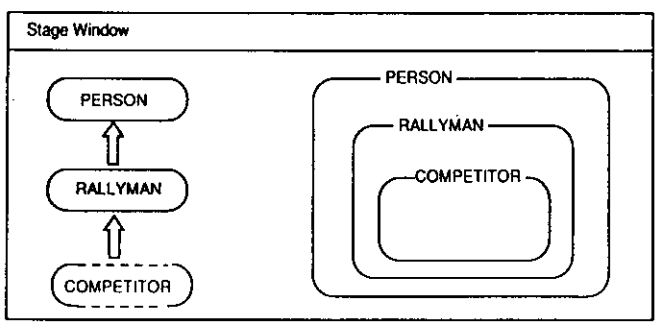
2. If T is selected in the Stage Window and if the selection defines a multivalued path, the command has the following effects.
   - The ISA hierarchy is displayed on the request graph.
   - The Venn diagrams represent the sets of the objects of the type T and of its subtypes attached to one root object of the component.
3. If the T selection defines a monovalued path, only the ISA hierarchy is displayed.

Example 8: In the following graph the command has been used on the PERSON node.



4. If T is selected in the Stage Window in an ISA network (successive uses of the command for the same type family), the ISA network and the Venn diagram relative to T are completed, respecting the semantics defined during previous uses of the command.

Example 9: Contents of the Stage Window after selecting RALLYMAN in the network of example 7



5.2.4 The ENTER Command

This command allows for entry of one or several mediatic objects (MO). A constant is specified (object or set constant) which can be used with other commands (COMPARISON).

The system displays a Venn diagram in which entry is carried out. Jokers can be used. For a MOT whose primary type is TEXT, the following jokers are available.

- * replaces a character string which may be empty
- ? replaces a character.

5.2.5 The DISPLAY Command

This command is used to specify the "objects for display" in the Result Window. These objects are marked by a small triangle. Naturally only MOT objects can be displayed. By convention, if this command is used on an AOT or a COT, the external identifiers will be displayed.

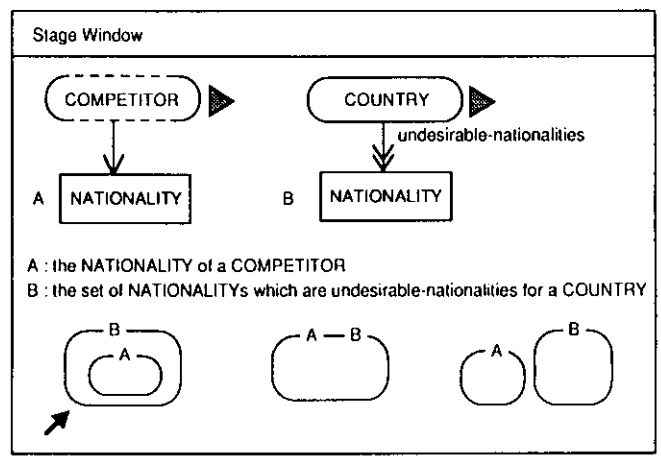
5.2.6 The COMPARISON Command

This command makes it possible to express a binary comparison. Two stages are necessary: the first for stipulating the operands t1 and t2 (the user proceeds by clicking in the Stage Window); the second for choosing an expression from the list proposed by the system.

Here the system plays an important role.

- It checks for the semantic compatibility of the operands.
- It displays their definitions in quasi natural language, after naming them if necessary.
- It proposes a list of comparison expressions appropriate to the nature of the operands (object or set) and to the kind of operand type (MOT, AOT, COT).

Example 10: We want to display the competitors who may be forced back across a border (we assume that the RALLY schema contains the type COUNTRY).



In this example, the operands are variables and an object-set comparison is specified. This request corresponds to the implicit definition of an object constructed by an aggregation between COMPETITOR and COUNTRY.

An operand can be defined in different ways:

- from a component by selecting a node;
- from Venn diagrams by clicking a zone or by using the COMBINATION command;
- directly by entering its value with the ENTER command.

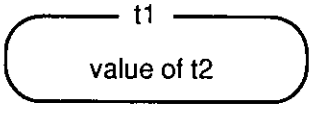
An operand is a constant if its value is fixed and does not depend on the object to search; otherwise it is a variable. A constant is said to be defined in extension if its value has

been entered with the ENTER command; otherwise it is defined in intention. A paraphrasing is proposed by the system for the different operands introduced by the user.

A comparison can be of one of three types:

- Type 1: t1 is a variable and t2 is a constant defined in extension.
- Type 2: t1 is a variable and t2 is a constant defined in intention.
- Type 3: t1 and t2 are variables.

As soon as the user has selected one of the comparisons, only this choice remains on the screen. The following summary remains on the screen when t2 is a constant defined in extension and the choice t1= t2 is selected:



The use of the COMPARISON command is limited during a stage. For each stage, it is possible to express simultaneously several comparisons of type 1 and only one comparison of type 2 or type 3. When a stage comprises several comparisons, the interface supposes that the corresponding boolean conditions are linked by logical ANDs. This interpretation corresponds to the usual comprehension in natural language.

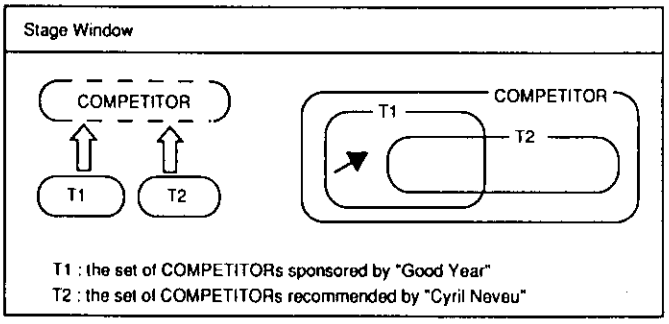
5.2.7 The COMBINATION Command

This command permits the definition of a set by set operations carried out on other sets previously defined. This is simply achieved by clicking on Venn diagrams (obtained, for example, with the STIPULATE command).

This command is useful for two reasons.

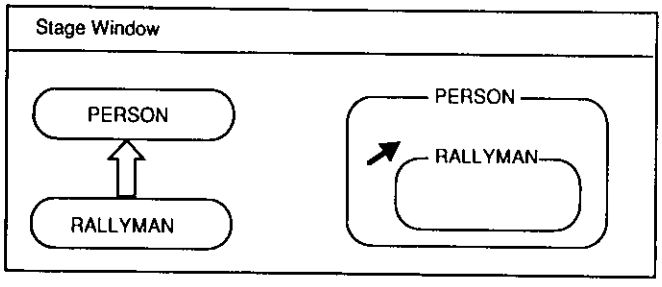
- It makes it possible to replace boolean operators, whose ambiguity for the user has often been noted, by clicking on Venn diagrams. It can therefore be used to combine the results of several stages. Indeed, at each stage, a simple derivation predicate p(T) can be expressed. The command therefore is equivalent to a boolean combination of simple predicates.
- Sometimes it is the only means for defining a subtype when there is no property on the attributes which might characterize it.

Example 11: We can imagine that two subtypes T1 and T2 have been defined for COMPETITOR during previous stages.



The clicking permits the creation of the set of COMPETITORs which are included into T1 and not included into T2. A new subtype is created in the window for future uses.

Example 12:



Here we are creating the subtype of PERSONs who are not rallymen.

5.2.8 Other Commands in the Stage Window

FUNCTION. This command makes it possible to evaluate a set of objects: type class or value of a multivalued attribute. The system proposes a list of evaluators appropriate to the kind of object type. The evaluator NUMBER (NUMBER OF OBJECTS) is always proposed. The MOTs have specific evaluators: MAXIMUM, MINIMUM, AVERAGE for numbers; THE LONGEST, THE SHORTEST for character strings.

EXPRESSION. This command makes it possible to specify an expression combining MOs of the same primary type. The list of operators depends on the primary MOT (+, -, for numbers; CONCATENATION for character strings).

GROUPING, AGGREGATION, HIERARCHIZATION. Explicit derivation commands for new COTs.

EXTENSION. A command for explicit derivation of a new attribute.

DELETE. This command deletes the selected object (type, attribute, Venn diagram) which allows for clarification of the Stage Window contents.

CANCEL. Cancels the stage being specified.

END. Indicates to the system the end of the stage definition.

5.3 Recapitulation Window Commands

EXECUTE: Starts the execution of the request paraphrased in the Recapitulation Window.

INSERT: Permits the insertion of a new stage in the stage sequence of a request.

SAVE: Stores the current request with all its stages.

LOAD: Recalls a request.

ADD: Adds the derived element selected in the Recapitulation Window to the external schema of the Schema Window. Almost all of the request (except information concerning display) can be included in the schema.

5.4 Result Window Commands

FORMAT: This command enables the user to choose the editing format from the different possibilities detected by the system.

5.5 General Commands

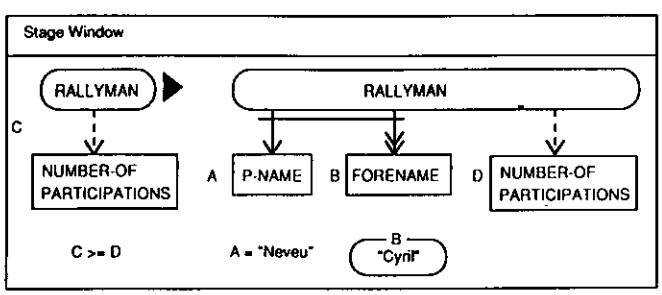
ENLARGE, REDUCE: These commands modify the window size.

END: Indicates the end of the session.

6. REQUEST EXAMPLES

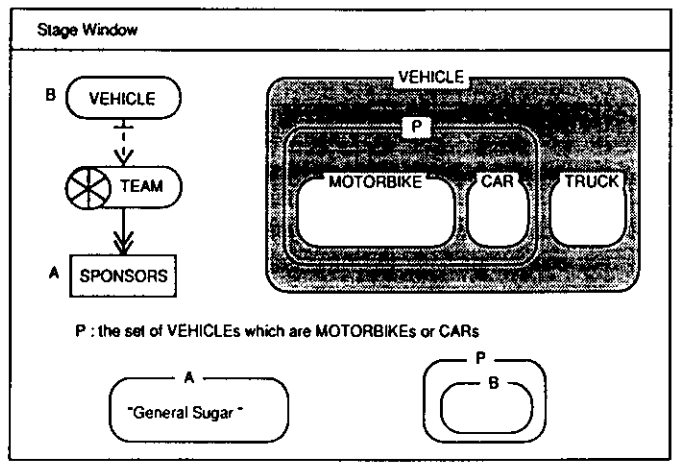
For each of these examples, only the simplified contents of the Stage Window at the end of the stage are represented.

Request 1: Which rallymen have participated as often as Cyril Neveu?



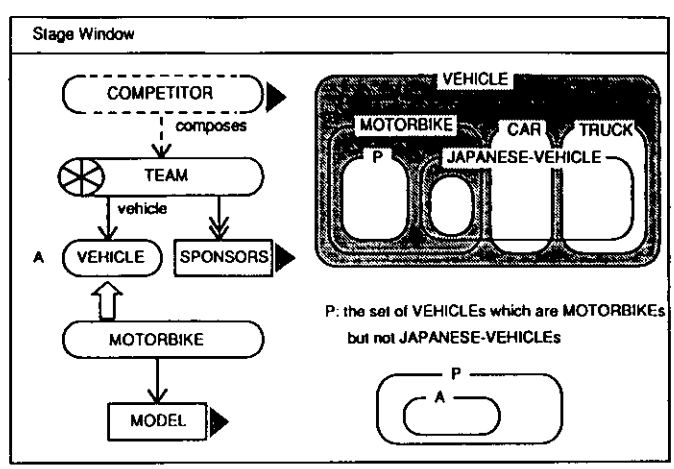
The graph has two components. The one on the left describes an object to be searched and the one on the right describes a constant used as operand. Each component has been created with the DEVELOP command. The ENTER command has been used twice (once for the name "Neveu" and again for the forename "Cyril") and the COMPARISON command has been used three times.

Request 2: Which motorbikes and cars are sponsored only by General Sugars?



The STIPULATE command has been used for selecting VEHICLE in the Stage Window. The set P has been created with the COMBINATION command. This example shows how quantifiers are expressed through Venn diagram combinations. It also shows how CANDID specifies the difficult case for the user where an "and" in natural language must be expressed by an "or" in the inquiry language. Note that this request could also have been expressed from a component constructed with the TEAM node as the root.

Request 3: Who are the competitors participating with a non-Japanese motorbike? For each one, specify the sponsors and the model of motorbike (we suppose here that the subtype JAPANESE-VEHICLE has been defined during a previous stage).



The STIPULATE command has been used twice: first on VEHICLE in the Stage Window to complete the component, then on VEHICLE in the Schema Window in order to be able to define P. The command DEVELOP subsequently allows a description of MOTORBIKE to be obtained, which was used for the command DISPLAY.

7. CONCLUSION

CANDID is a candidate system in the new generation of graphical semantic interfaces and so it is important to compare it to the two major systems of this generation: ISIS and SNAP.

ISIS (Goldman et al. 1985) is based on a subset of the SDM model (Hammer and McLeod 1981). It offers an original graphical representation of the schema. Links between types are represented through a forest. Attributes are defined through visual references with color patterns. A request is specified in an interactive manner by constructing derived elements. The request language is essentially textual in spite of the fact that there is an interesting graphical representation of the boolean expressions. In relation to CANDID, the graphical representation of a request is therefore not available and that of the schema is less complete.

SNAP (Bryce and Hull 1986) is based on a subset of the model IFO (Abiteboul and Hull 1987). The graphical representation of the schema uses the notion of "fragments" introduced with the model. Distinct roles of an object type lead to distinct representations, thus allowing for modular viewing of the schema. A graphical language makes it possible to construct the request graph. Interesting possibilities for data formatting are offered. In relation to CANDID, the version of SNAP presented in Bryce and Hull does not allow for the expression of boolean operators, or the incorporation of derived elements into the schema. Nevertheless, the schema graph can be represented in a unified way.

The CANDID interface is based on a generic semantic model (Schneider and Trépied 1989) and so it offers more freedom to the user during inquiry. A request can be expressed in different ways: the stage sequence is not unique and with each stage several different graphs are possible. Almost all of the request (all the derived elements) can be incorporated into the schema. Moreover, using Venn diagrams to express the boolean operators and the quantifiers makes the work of the final user easier and improves its quality (Michard 1982; Harel 1988). The systematic paraphrasing of specified elements is an attempt to help the user in reducing the number of erroneous requests.

Several points are being investigated further, particularly the power and the completeness of the language. The study of the power necessitates a formal definition which is presently under study. The completeness is difficult to characterize since there is no universal definition for complex objects (Abiteboul and Grumbach 1987).

The implementation is being carried out on an APOLLO workstation. Our aim is to make CANDID easily adaptable to various contexts (object oriented DBMS, relational DBMS offering deduction mechanisms). For this purpose, the software has been designed in a modular and parameterized way. Its architecture makes it possible to extend the model (for example, with new construction relationships) and to modify the graphical symbolism as in the ZOO system (Riekert 1986).

8. REFERENCES

Abiteboul, S., and Hull, R. "IFO: A Formal Semantic Database Model." *ACM TODS*, Volume 12, Number 4, December 1987, pp. 525-565.

Abiteboul, S., and Grumbach, S. "Bases de Données et Objets Structurés." *Théorie et Science Informatique*, Volume 6, Number 5, 1987, pp. 383-404.

Banerjee, J.; Kim, W.; and Kim, K. C. "Queries in Object-Oriented Databases." *Fourth International Conference on Data Engineering*, Los Angeles, 1988, pp. 31-38.

Bryce, D., and Hull, R. "SNAP: A Graphics-Based Schema Manager." *Proceedings of the 2nd IEEE International Conference of Data Engineering*, New York, February 1986, pp. 151-164.

Corson, Y. "Ergonomie des Langages de Requéte Relationnels." *Théorie et Science Informatique*, Volume 2, Number 5, 1983, pp. 329-339.

Goldman, K. T.; Goldman, S. A.; Kanellakis, P. C.; and Zdonik, S. B. "ISIS: Interface for a Semantic Information System." *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, New York, 1985.

Hammer, M., and McLeod, D. "Database Description with SDM: A Semantic Database Model." *ACM TODS*, Volume 6, Number 3, September 1981, pp. 351-386.

Harel, D. "On Visual Formalisms." *Communications of the ACM*, Volume 31, Number 5, May 1988, pp. 514-530.

Heiler, S., and Rosenthal, A. "G. WHIZ, A Visual Interface for the Functional Model with Recursion." *Proceedings of VLDB Conference*, Stockholm, 1985, pp. 209-218.

Hull, R., and King, R. "Semantic Database Modeling: Survey, Applications, and Research Issues." *ACM TODS*, Volume 19, Number 3, September 1987, pp. 201-260.

Kim, H.; Korth, H. F.; and Silberschatz, A. "PICASSO: A Graphical Query Language." *Software-Practice and Experience*, Volume 18, Number 3, March 1988, pp. 169-203.

Michard, A. "A New Database Query Language for Non-Professional Users: Design Principles and Ergonomic Evaluation." *Rapport de Recherche Number 127*, INRIA, April 1982.

Miranda, S., and Nsonde, J. "LAGRIF: A Pictorial Non-Programmer-Oriented Request Language for a Relational Database Management System." *International Conference on Improving Database Usability and Responsiveness*, Jerusalem, June 1982, pp. 173-204.

Pauthe, P. "EVER: Un Éditeur de V-relations." *Thése de 3éme Cycle*, Number 3957, Université de Paris-Sud, September 1985.

Potter, W. D., and Trueblood, R. P. "Traditional, Semantic, and Hyper-Semantic Approaches to Data Modeling." *Computer*, June 1988, pp. 53-63.

Riekert, W. F. "The ZOO Metasystem: A Direct Manipulation Interface to Object-Oriented Knowledge Bases." Institut fér Informatik, Universitét Stuttgart, 1986.

Schneider, M., and Trépied, C. "Un Modéle Sémantique pour la Manipulation de Base de Données par L'utilisateur Final." *Rapport Interne*, Laboratoire d'Informatique, Université Blaise Pascal de Clermont-Ferrand, March 1989.

Tanaka, K.; Yoshikawa, M.; and Ishihara, K. "Schema Virtualization in Object-Oriented Database." *Fourth International Conference on Data Engineering*, Los Angeles, February 1988, pp. 23-30.

Zloof, M. M. "Query-By-Example: A Database Language." *IBM Systems Journal*, Volume 16, Number 4, 1977, pp. 324-343.