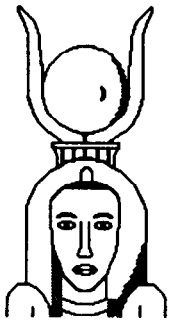

ISIS:

Interface for a Semantic Information System

Kenneth J Goldman Sally A Goldman
Paris C Kanellakis[1] Stanley B Zdonik

Brown University

Abstract

ISIS is an experimental system for graphically manipulating a database The system is based on a simply specified high-level semantic data model It demonstrates the capabilities of a workstation environment by integrating three aspects of database programming in one graphical setting Namely, it permits database construction and modification, it allows browsing at the schema and data levels, and provides a graphical query language In all of these activities it maintains uniform graphical representations and consistent user interaction techniques

1. Introduction

ISIS is a system that exploits the visual dimension for database programming It allows users to construct, maintain, and query a database using a graphics interface and a consistent operational paradigm Based on a high-level semantic data model, ISIS is an experiment in integrating several forms of database programming into a single interface that is rich in capability yet intuitive enough for non-experts to use

The construction of database retrievals constitutes a very important part of programming in commercial data processing A system like ISIS allows a broad class of users to become "database programmers" and can substantially reduce the amount of time required to construct programs of this type

1.1. Visual Query Languages

Many of the database query languages that have appeared suffer from the fact that they are textually oriented and very formal Although simple queries are reasonably straightforward, slightly more complex queries exceed the capabilities of a novice user The use of the visual dimension seems to hold promise as a way of providing a more intuitive interface in the context of a two-dimensional syntax

ISIS uses the visual dimension to integrate three aspects of database programming With ISIS, a user is able to build a databasse or modify an existing one, to browse through the contents of a database in order to answer questions about the data or the schema, and to construct queries that can be saved for later use All of these activities are accomplished using the same style of interface and the same iconic representations, so that a user is able to move easily from one activity to another at any time

Other efforts [Zl, MS, He, Ki] have made some progress in this direction Query by Example (QBE) [Zl] is a relational query language that allows a user to fill example values into templates of relations The system then determines which

[1] On leave from Brown University, current address Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Mass 02139

This research was supported in part by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014-83-K-0146 and ARPA Order No 4786

tuples satisfy this pattern and prints the specified results Cupid [MS] is a graphical query facility to a relational database that allows users to construct a two-dimensional picture that can be interpreted as a query The system represents a relation with a picture of a representative tuple from that relation (one slot per field) One draws labeled arcs between fields to indicate the constraints that should hold on the answer The arcs are labeled with comparison operators SDMS [He] is a very highly-developed browser that allows a user to navigate with a joy stick through a space of icons that represent the entities in the database It is possible to zoom in on an entity at any time to obtain more details of that entity SKI [Ki] is a system that allows a user to build a query graphically Like ISIS, it is based on a semantic data model, however, unlike ISIS, it does not provide the integrated facilites for schema construction and for browsing at the entity level A number of browsers using high powered graphics, but with limited query capabilities have also been developed [Fo,Ca,SK]

1.2. Semantic Data Models

The relational data model [Co1] has recently achieved a great deal of popularity This is largely due to the simplicity and uniformity of the model It is easy to learn and has successfully isolated semantic issues from implementation concerns At the same time it has been recognized that the relational model does not have sufficient expressive power to specify directly some of the complex data relationships that one encounters in applications modeling Recently, a great deal of work has been done on the development of higher-level data models that have more expressive capability than pure relations, [BF, BN, Ch, Co2, HM, HY, MBW, S, SS, TL, WMy, Zd] are only some of the efforts in this direction

The Semantic Data Model (SDM) [HM] is one such high-level model, and it provides the underpinnings for this work ISIS supports a graphical interface to a modified subset of the features of the SDM We will informally summarize the main features of the SDM in this section and describe the ISIS subset in the section that follows

Users create and manipulate *entities* when using an SDM database An entity corresponds to anything in the application that is semantically meaningful The most central concept in the SDM is that of a *class* A class is a collection of entities, all of a similar type Entities have associated *attributes* An attribute is defined to have a *name* and a *value class* A value class is some class in the SDM database from which the values of the attribute are drawn All members of a class have a common set of attributes

Classes can be related to each other via *interclass connections* The two most common interclass connections are *subclass connections* and *grouping connections* A subclass connection indicates a relationship between some class S and a class T that is constrained to contain a subset of the elements of S T is said to be a *subclass* of S, and S is said to be a *superclass* of T The subset can be defined by enumeration of the members, or by a predicate such that T contains exactly those members of S that satisfy the predicate Some classes are distinguished as *baseclasses* A baseclass is one that has no superclasses A grouping connection is one that relates a class S to a grouping class T, where T contains sets of entities from S as members

Members of a class are said to *inherit* the attributes from all of their superclasses That is to say, if T is a subclass of S, all members of T will automatically be defined to have all those attributes that are defined on S (as well as the attributes of all the superclasses of S)

In order to make the construction of the initial experimental version of ISIS more tractable, we have selected a modified subset of the SDM as our data model For this subset, we chose those features that would make our system relationally complete and useful One major point of departure is the way that the our data model handles groupings In ISIS a grouping is only allowed on common values of an attribute Another is that we limit the inheritance behavior of a subclass to single parent inheritance

It should be noted that ISIS views the construction of a query as equivalent to defining a new derived class The derived class is specified in terms of a predicate that corresponds to the query expression of more traditional languages ISIS allows users to build this predicate using graphical means

The basic semantic model constructs used are in Section 2 Their graphical representations are contained in Section 3 2 and an extended example is in Section 4 A measure of the success of the ISIS interface is the degree in which the example of Section 4 suffices to demonstrate most of the system's non-trivial features

2. Basic Concepts

The basic concepts presented in this section are the building blocks of our data model, and are sufficient to describe most of the present capabilities of ISIS They correspond to essential features of existing semantic data models, and, in particular, reflect the basic design principles of SDM

Entity: An entity corresponds to an object in the application environment Each entity has a unique name, which is a string

Class: A class is a named set of entities The set of all entities is partitioned into disjoint classes called *baseclasses* If class C is not a baseclass, it is associated with a single other class *parent*(C), where C $\subseteq$ parent(C) We then say that C is a *subclass* of parent(C) There are four predefined baseclasses the *Integers*, the *Reals*, the *Booleans (Yes/No)*, and the *Strings*

Attribute: Let C, V be classes, then attribute A of C with *value class* V is a function from C to the subsets of V We say that A is *multivalued* (A C $\rightarrow\!\!\!\rightarrow$ V) unless this function is constrained to map each element of C to a singleton subset of V, then we say A is *singlevalued* (A C $\rightarrow$ V) In the singlevalued case, A defines a function from C to V

Map: Let x be an entity of C_1 and A_i $C_i \rightarrow C_{i+1}$, $1 \leq i \leq n$, then A_1A_2 $A_n(x) = \{ e \mid$ there exists a sequence of entities $x_1, x_2, \ldots, x_n, x_{n+1}$, such that, x_i is in C_i, $x = x_1$, $e = x_{n+1}$, and $x_{i+1} = A_i(x_i)$, $1 \leq n\}$ We call A_1A_2 A_n, $(n \geq 1)$ a map (*from* C_1 *to* C_{n+1}) For $n = 0$ we have the *identity map from* C *to* C (i e , x is mapped to {x})

Grouping: Let A be an attribute of a class C with value class V, then grouping G of C on A is the following family of subsets of C indexed by the members of V, G= $\{S_e \mid$ entity e in V, and entity x of C is in S_e if and only if e is in A(x)$\}$ We call class C the *parent*(G) Unlike classes, groupings have no attributes, subclasses or groupings However, we do want groupings to be ranges of attributes For this we allow attribute B to be a function from a class S to a grouping G This attribute B is treated as B S$\rightarrow\!\!\!\rightarrow$ parent(G)

Inheritance: Let class C be a subclass of parent(C), then every attribute A of parent(C) is also an attribute A of C Since C $\subseteq$ parent(C), a function defined on parent(C) has a natural restriction on C Inheritance of attributes is from the one parent class to the child class Note that a class could be a subset of another class without being its subclass, in this case, attributes are not inherited

Given a collection of classes, attributes, and groupings one can naturally define two directed graphs, whose nodes correspond to the given classes and groupings

The inheritance forest, with arc (X,Y) iff X = parent(Y) The inheritance forest is easily seen to be a collection of directed trees, where each tree contains exactly one baseclass node, its root A grouping node can only be a leaf in these trees

The semantic network, with arc (X,Y) labeled A iff A is attribute of class X with value class Y The semantic network is a standard construction we use a single arrow for singlevalued and a double one for multivalued attributes Note that in it no grouping node has outgoing arcs The outgoing arcs of a class node correspond to its attributes, including those that are inherited If a grouping node corresponds to a grouping on attribute A, we label it with A

Remark We limit our description to *single parent inheritance* This is because this type of inheritance combines a wide range of applictions with a single tree representation This is also for ease of exposition, the system is currently being extended to handle *multiple parent inheritance*

Schema: A schema is an inheritance forest and a semantic network on the same set of nodes These nodes are either class nodes or grouping nodes

Data: Let D be a schema, we associate
(1) a baseclass with each root of the inheritance forest,
(2) a class C with every class node, such that C $\subseteq$ parent(C),
(3) a (singlevalued) attribute with every (single arrow) arc of the semantic network,
(4) the grouping G on A of parent(G) with every grouping node labeled A

We assume that the standard baseclasses, *Integers*, *Booleans*, *Reals*, and *Strings*, are always in our schema and contain as data all integers, booleans, reals and strings of interest We also assume that entity names are determined by a special singlevalued naming attribute of each baseclass

Remark We have defined a syntactic notion, the schema, and a semantic notion, the data To guarantee an acceptable level of *integrity* we require that the data be *consistent* with the schema This notion of integrity represents a reasonable requirement we impose on the system at low computational cost

Data is consistent with the schema in the sense that each entity is in one baseclass only, each subclass is a subset of its parent, a singlevalued attribute defines a function, and each grouping is completely determined from its parent class and an attribute

We allow arbitrary modifications of the data and/or the schema, such as *insertions, deletions* and *updates*, as long as the data remains consistent with the schema For example, in the data level, we can insert an entity in a class, provided we also insert it in its parent and specify a value for its naming attribute If we do not specify a value for any other singlevalued attribute, the default is the *null* entity, which we assume to be a member of every class If we do not specify a value for a multivalued attribute of an entity the default is the empty set In the schema level, we may delete a class, provided it is not the parent of some other class or the value class of some attribute

The insert, delete, and update facilities are simple but do not provide for querying the database In order to build queries we use

Derived Subclasses: Let V be a class in the schema A derived class S can be defined from V using a predicate P on the entites of class V, which becomes parent(S) $S = \{ e \mid e \text{ in } V \text{ and } P(e) = \text{true}\}$

Derived Attributes: Let V,C be classes in the schema A derived attribute A can be defined from C to V If x is an entity in C, A(x) is defined using a predicate P_x on the entities of V Formally, for x in C, $A(x) = \{ e \mid e \text{ in } V \text{ and } P_x(e) = \text{true}\}$

Predicates P(e) and $P_x(e)$, x in C, e in V, can be constructed from *atoms* using the boolean connectives **and, or.** The atoms of P(e) are of the form (a) or (b), and the atoms of $P_x(e)$ are of the form (a), (b) or (c)

(a) $<mapv_1(e)> <operator> <mapv_2(e)>$

(b) $<mapv(e)> <operator> <mapc(w)>$, $w \subseteq C$

(c) $<mapv(e)> <operator> <mapc(x)>$

The *mapv* and *mapc* are maps from classes V,C respectively (they could be the identity maps) Set comparison *operators* used are set equality (=), subset and superset operators ($\subseteq, \supseteq, \subset, \supset$), and a weak match operator ($\asymp$) to determine if two sets have a common element In addition, ordering operators ($\leq, \geq$) are available for comparing singleton sets The negations of all these operators are also available Finally, there is a shorthand unary operator (represented by the hand icon) for assigning some $<mapv(e)>$ to be the derivation of an attribute These predicates provide the full power of relational algebra, and other operators can be easily added to enhance data manipulation capabilities

Derived subclasses and attributes are examples of schema and data modifications that do not violate any consistency requirements We transfrom old data, consistent with an old schema, into new data, consistent with a new schema However, the predicates of derived subclasses and attributes do not (at present) form part of the consistency requirements of the system

3. System Description

In this section we describe the various ISIS features and capabilities ISIS provides multiple views of the database schema, as well as views of the data itself

A *view* corresponds to an entire workstation screen A view could contain (1) *menus*, (2) *text-windows*, and/or (3) *windows* All of these are disjoint rectangular areas within the view

Menus are standardized commands for each view These commands are consistent within the various views That is, commands in different views with the same names have the same semantics Selection of the menu commands is made with a one-button mouse and, in some cases, can also be made through function keys This last deviation from mouse purity is a simple convenience, which greatly speeds up interaction

Text windows are areas used for textual input (from the keyboard) and for textual ouput A text window can be used for (1) system error warnings, (2) system prompts for user input from the keyboard, mouse, or function buttons, and (3) system text ouput

Windows are areas containing graphical representations of subsets of the schema or the data The metaphor here is that the graphical representations of the schema and the data exist in their respective *planes* The windows show us a piece of these planes, some of the decisions for window positioning over the planes are made automatically by the system Commands are always provided for manually changing the window position (e g , *panning* commands) Where appropriate, a *graphical editor* is provided for changing the graphical representations The one-button mouse is used for selecting rectangular areas inside the windows

3.1. A Two-Level Approach

ISIS operates at two levels, the *schema level* and the *data level*. Diagram 1 illustrates how one moves among these levels during a database session. The schema level provides views of the schema plane. These views are the *semantic network*, the *inheritance forest*, and the *predicate worksheet*. The data level provides views of the data plane.

In both levels, *navigation* is possible using the maps formed by attributes in the schema. The state of ISIS consists of a *schema selection* (the class, attribute, or grouping being examined) and a *data selection*. Schema selection can be changed (S ← S') at both levels as part of navigating through the schema. Data selection can be changed (D ← D') at the data level. When one switches levels temporarily to select a constant or create a user-defined subclass (see loop arrows in Diagram 1), neither the schema selection nor the data selection are changed upon returning from the temporary visit to the other level.

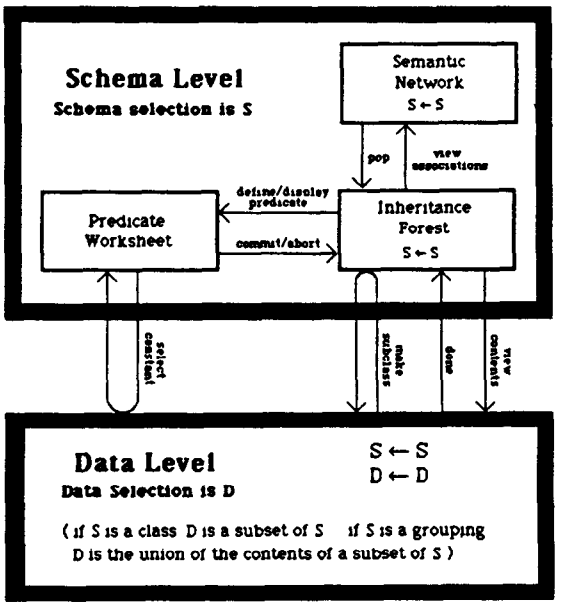

Diagram 1 Interconnections of ISIS components

3.2. Graphical Representations

Classes have three parts (1) a class name section, for baseclasses this is in reverse video, (2) a characteristic fill pattern unique to the class, which is provided automatically by the system, and (3) an attribute section containing a number of attributes.

Attributes, in the class attribute sections, contain their name and the fill pattern of their value class. If an attribute is multivalued, this fill pattern is shown with a white border to signify that the attribute value is a set. The first attribute in a baseclass is the naming attribute.

Maps are represented by a stack of classes. These are the classes linked by the map attributes.

Groupings are represented in the same way as classes, but they have no attribute sections and their characteristic fill patterns have a white border to signify that their members are sets.

Inheritance of attributes is explicitly represented in one view, and implicitly in the others, through their automatic addition to the attribute section of a class.

Schema Level:

In the inheritance forest view (see Figure 1), lines connect parent classes to their children and the system enforces some of the placement decisions. Namely, groupings always appear above their parent class and subclasses below. In this view classes do not contain inherited attributes, which appear automatically in all other views. A hand icon is used to point to the schema selection. An editing menu is available at the right for panning within the view, moving classes and groupings, deleting classes, attributes and groupings, and undoing and redoing actions.

The inheritance forest view provides variants of a menu of commands for (re)naming the selection, going to the semantic network view (view associations), going to the predicate worksheet (define), and going to the data level (view contents). Schema selection is changed by picking some other object with the mouse. The commands on the menu vary according to whether the schema selection is a class, an attribute or a grouping. For example if the schema selection is a class then a subclass or attribute can be created. If the schema selection is an attribute then a list of all classes can be created, as a pop-up menu, for selecting the value class. This selection can also be made with the mouse in the view.

An alternate view at the schema level, the semantic network, consists of one window, in which there are classes, groupings, and arcs as defined in section 2 (see Figure 2). The semantic network may be used for navigation in the schema. At any point one may pop back to the inheritance forest view with the new schema selection.

The predicate worksheet consists of several windows (see Figure 9) The atom construction window at the lower right contains three subwindows for the left hand side, the operator, and the right hand side Maps are specified by choosing the map attributes with the mouse and forming a stack of classes The four right hand side options are **map**, for constructing a map from the entity, as on the left hand side, **map starting at class**, for contructing a map starting at an arbitrary class chosen from the class list window on the right, **constant**, which temporarily takes the user into the data level, where he may select or create a constant in the class at which the left hand side mapping terminates, **constant starting at class**, which allows the user to select the class (from the class list window) in which he would like to start searching for the constant at the data level As atoms are being constructed, feedback is provided above the atom creation window in the atom list window, which contains atoms that have been constructed or are being constructed These atoms may be edited and placed in clauses (the set of windows on the left) in disjunctive or conjunctive normal form

Data Level:

The view here contains a number of overlapping pages (see Figure 3) The top page contains the schema selection, a class or grouping and the data selection, some of its members Each page contains a class, with all of its attributes including inherited ones, or a grouping To the right of each class or grouping is a pannable list of its members Selected members are highlighted with bold text Navigation is possible at the data level by following attributes It is also possible to pop backwards At the data level one can change the schema selection, assign/modify attribute values, and create new entities and new classes The latter is performed by manually selecting entities and using the make subclass menu command to temporarily visit the inheritance forest to name and position the new class Returning to the inheritance forest correctly sets the hand icon pointing at the new schema selection

4. Example

The following example illustrates the major functionality of the interface We assume an existing database (schema plus data), described in section 4 1 Then, in section 4 2, we describe an ISIS session that makes use of the existing database Although the example does not illustrate building a database from the beginning, the techniques used in section 4 2 for adding to and modifying the database may be used equally well for schema definition and data entry

4.1. Sample Schema

The database *Instrumental_Music* has baseclasses *musicians*, *instruments*, *music_groups*, and *families* (see Figure 1)

The *musicians* baseclass has three attributes *stage_name*, providing names for the entities, *plays*, which is a multivalued attribute with the value class *instruments*, and is the set of instruments that each musician plays, and *union*, which maps into the *YES/NO* class and indicates whether or not the given musician belongs to the musicians' union The groupings *by_instrument* and *work_status* group the entities in *musicians* according to the instruments they play and according to whether or not they are union members The subclass *play_strings* contains those musicians who play at least one intrument whose attribute *family* has the value stringed This subclass has an attribute, *in_group*, which maps into the *YES/NO* class and indicates whether or not the string player is the value of the *members* attribute of some entity in the class *music_groups* The *play_strings* subclass also has an associated grouping, *by_in_group*, which groups musicians on the basis of the attribute *in_group* The subclass *soloists* is user-defined (i e , formed by hand-picking entities from the parent class

The baseclass *music_groups* has four attributes *name*, the name of the music group, *members*, a multivalued attribute, which maps into the class *musicians* and represents the members of the given music group, *size*, which has the value class *INTEGER* and is the number of members in the group, and *includes*, which maps into the class *families* and contains the families of instruments that are played by the music group

The baseclass *instruments* has three attributes *name*, the name of the given instrument, *family*, which maps into the class *families* and is the family of the instrument, and *popular*, an attribute mapping into the *YES/NO* class The grouping *by_family* partitions the instruments into sets according to their *family* attribute

The baseclass *families* contains types of instruments (e g , brass) and has attribute *name*

4.2. Sample Session

The user desires to find entertainment for a department holiday party Upon entering ISIS, he

loads the database *Instrumental_Music* and begins by familarizing himself with the database Upon loading the database, he sees the inheritance forest view of the database Since there is no schema selection, he chooses an object on which to focus his attention With entertainment in mind, he moves the cursor to the class *soloists* and clicks the mouse button (this action will be referred to as **picking**) *Soloists* becomes the schema selection, identified by the hand icon as in Figure 1

In order to find out more about the information associated with soloists, he chooses the function button, **view associations** (Note Although functions need not be picked with the mouse, example figures often show the cursor on the chosen function key description for clarity) Upon examining the semantic network for soloists, the user becomes interested in the multivalued attribute *plays* and picks its value class, *instruments* This causes the *instruments* class to become the new schema selection and its semantic network to be displayed as in Figure 2

Deciding that he wants to see the contents of *instruments*, he picks the **pop** button to return to the inheritance forest, where *instruments* is now the schema selection, and then presses the function button, **view contents**, which takes him into the data level He now sees *instruments* with all its attributes and a list of its member entities Using **select/reject**, he can choose members on which to focus his attention, these are highlighted with a large boldface type In Figure 3, he has already selected the flute and is selecting the oboe

Next he wants to find the families associated with these entities Upon pressing the **follow** function key, he is prompted to choose an attribute, and picks *family* The selected entities from *instruments* are listed in boldface type under that class, and an arrow is drawn from the followed attribute to its value class (Figure 4) Since brass is the only family highlighted on the new page, it appears that both flute and oboe are in the brass family But the user knows that these instruments are both woodwinds and decides to correct the error Using **select/reject**, he unhighlights brass and highlights woodwind He then uses **(re)assign att. value** to update the family attribute for both flute and oboe simultaneously (Figure 5)

Deciding to find out more about families, the user returns to the inheritance forest and changes the schema selection from *families* to *by_family* Wondering if this is related to the class *families*, he uses **display predicate** and finds that this grouping indeed contains sets of instruments grouped by common value of their *family* attribute He returns to the entity level in this grouping class and selects the percussion family (Figure 6) By pressing the **follow** function key he sees all instruments with the percussion instruments highlighted (Figure 7) When **follow** is applied to a grouping, obviously no attribute selection is required, we merely follow the selected set(s) into the parent class and highlight the members of the set(s)

Satisfied that he has browsed enough to familiarize himself with the database, he returns to the schema level to begin his query Rather than randomly selecting a music group, he decides to form a subclass of *music_groups* that will satisfy his requirements He desires a music group of four musicians because of the size of the gathering He also has the special requirement that at least one of the musicians must play the piano, since a friend of his plans to sing at the party and will need an accompanist After pressing the **create subclass** function key, he is presented with a box which he may drag into position (Figure 8) He names this new class *quartets* using **(re)name**

He then selects **(re)define membership**, which takes him into the predicate worksheet He begins by selecting atom A, which will specify that the size of the group is four. He puts this atom into the second clause and picks the **edit** button He then picks the attribute *size* on the left hand side of the atom creation window and the predefined class *INTEGER* is added to the stack of classes on the left (since the attribute *size* maps into *INTEGER*) After picking the equality operator, he proceeds to the right hand side, where he picks **constant** Since the left hand side map terminates in the *INTEGER* class, he is taken temporarily into the data level with the *INTEGER* class showing After selecting the constant {4}, the user is returned to the predicate worksheet Next, he edits atom E, which will require that at least one musician in the quartet plays the piano He puts atom E in the first clause, specifies the left hand side map, and chooses the superset operator (Figure 9) After selecting the constant {piano} and changing the predicate to conjunctive normal form with the **switch and/or** button, he presses **commit**, which causes evaluation of the predicate and his return to the inheritance forest

Now the user would like to know all instruments played by the members of each music group in the *quartets* subclass To accomplish this, he creates an new attribute, names it *all_inst*, and then uses **(re)specify value class** to tell the

system that the values of this attribute come from the class *instruments* He then chooses **(re)define derivation** except for the addition of the unary assignment operator (shown as a hand icon), he sees the same predicate worksheet as before Using the same mechanisms as before, he specifies the attribute derivation, which is shown completed in Figure 10

After returning to the inheritance forest, the user is ready to look at the entites in the *quartets* subclass and enters the data level Finding only one quartet has met his requirements, he decides to examine it more closely

He follows the attribute *members* to see who is in the quartet After seeing the members he thinks that Edith sounds familiar and wants to focus on her Therefore, he uses **select/reject** to unhighlight the other members (Figure 11) He follows *plays* and sees that she plays the viola and the violin The user wants to remember this information so he presses **make subclass**, which takes him temporarily to the inheritance forest, where he names this new subclass *edith_plays* and positions it under *instruments* (Note this subclass automatically becomes the child of the class on the current page at the data level) After more browsing, he is satisfied and returns to the inheritance forest, where he sees the subclass that he created (Figure 12)

Finally, he feels that he has seen enough for now and picks **stop** In case he needs to come back and browse some more, he saves this new database as *entertainment*, and then phones LaBelle Musique

In this session, the user was able to become familiar with the organization of a database, through browsing in both the schema level and the data level As he browsed, he was able to modify the data to correct an error He then added to the schema in forming a query, he created a new subclass, and then added a derived attribute to that subclass Finally, he was able to explore the result of his query at the data level and to create a user-defined subclass containing some information that he wanted to remember

5. Summary

We have described an experimental system for graphically manipulating a database This system integrates several aspects of database programming In particular, it allows users to construct schemas, to browse through the database at both the schema and the data level, and to formulate queries that can be stored as part of the schema and reused at some time in the future

We feel that this system builds very strongly on previous work and contributes a paradigm of interaction that integrates more functionality than any of its predecessors This integration is achieved by using a uniform model of data coupled with a simple set of functions and a single iconic representation

ISIS is a part of a larger effort at Brown University to build a programming environment based on visualizations of the structures that are required to create programs This effort stresses the need to view these program structures in multiple ways and to make changes to the underlying programs by directly manipulating the graphical images Other components of this environment include PECAN [Re] and BALSA [BS] ISIS is implemented on an Apollo workstation in the C programming environment It uses the ASH graphics package and the APIO input package, both developed at Brown

At this point, ISIS is only an experimental vehicle We see several interesting directions in which this work could proceed First, we would like to include additional SDM features into our system without disturbing the smoothness of the interface For example, we are working on providing multiple inheritance Second, we would like to be able to specify arbitrarily complex predicates in a similar graphical way as a part of an integrity constraint specification system For example, how would a user specify that an employee cannot earn more than his/her manager using only a screen and a pointing device? Third, we would like to add features to assist users in the process of designing their schemas, as in [RBBCFKLR] For example, it would be useful to be able to keep track of the history of a database design

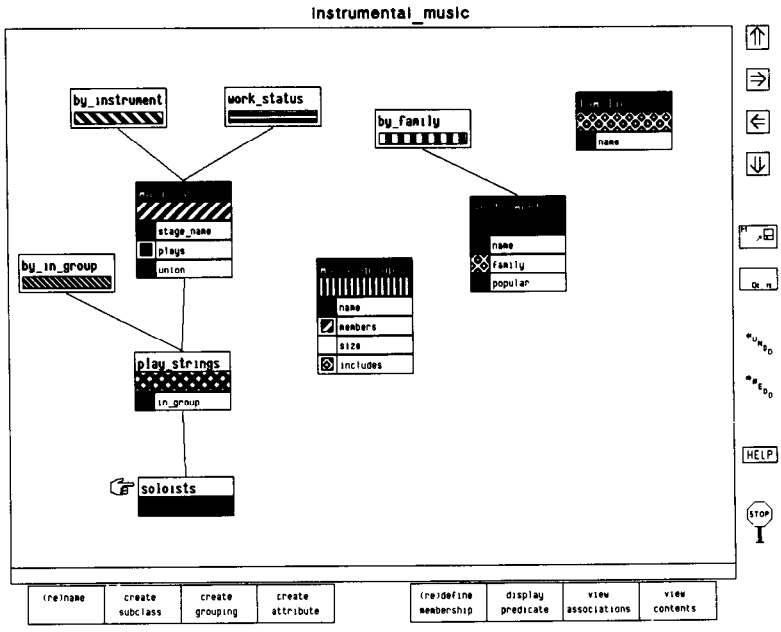

Figure 1. The inheritance forest view with *soloists* as the schema selection

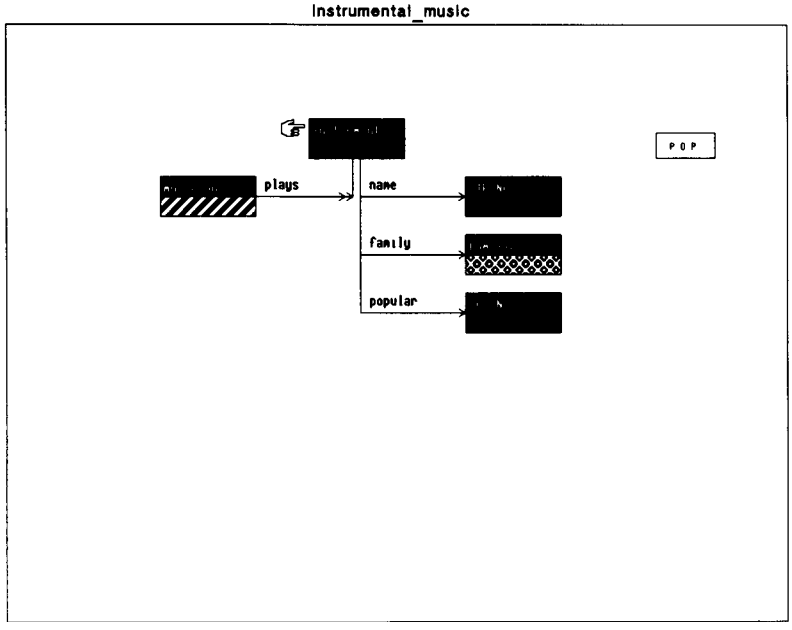

Figure 2. The semantic network view with *instruments* as the schema selection

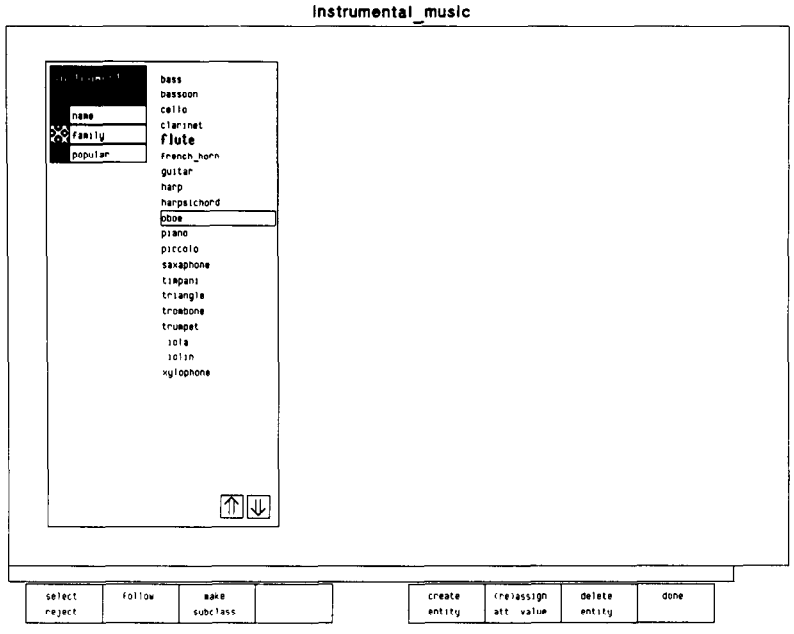

Figure 3. Selecting the entity *oboe* from the *instruments* class at the data level

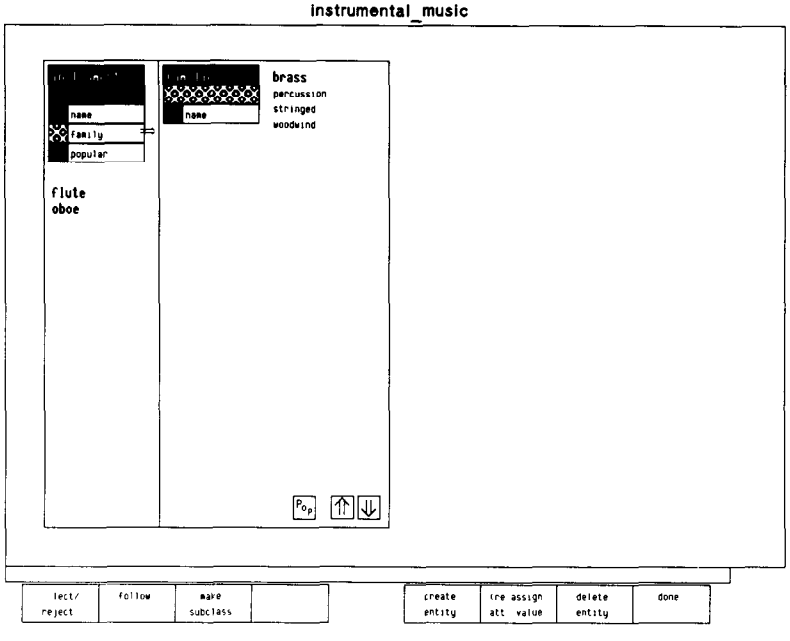

Figure 4. After following the *family* attribute for the entities *flute* and *oboe*

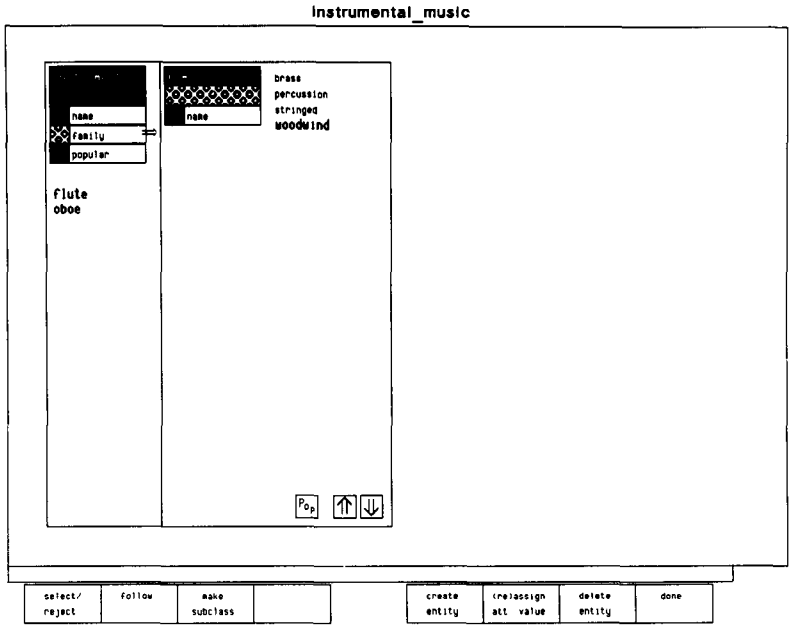

Figure 5. Updating the *family* attribute for both *flute* and *oboe*

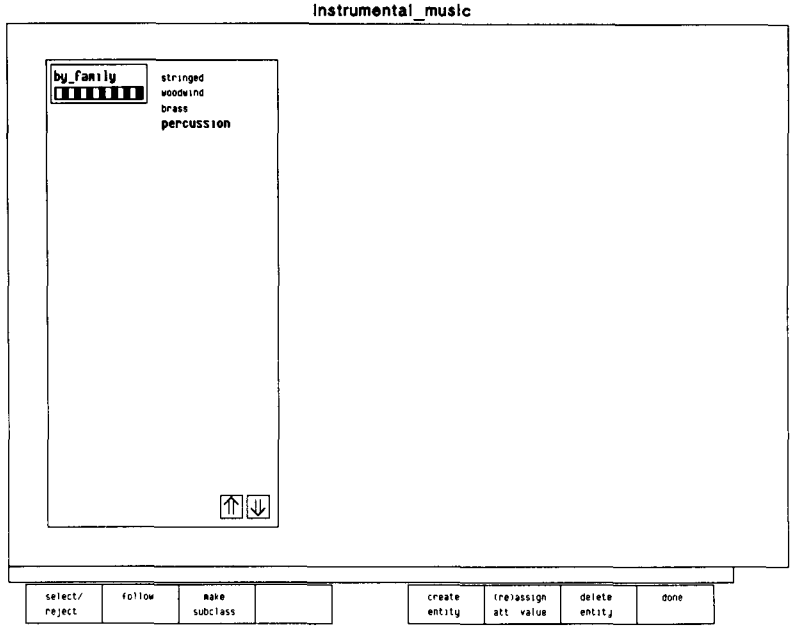

Figure 6. The *by_family* grouping at the data level

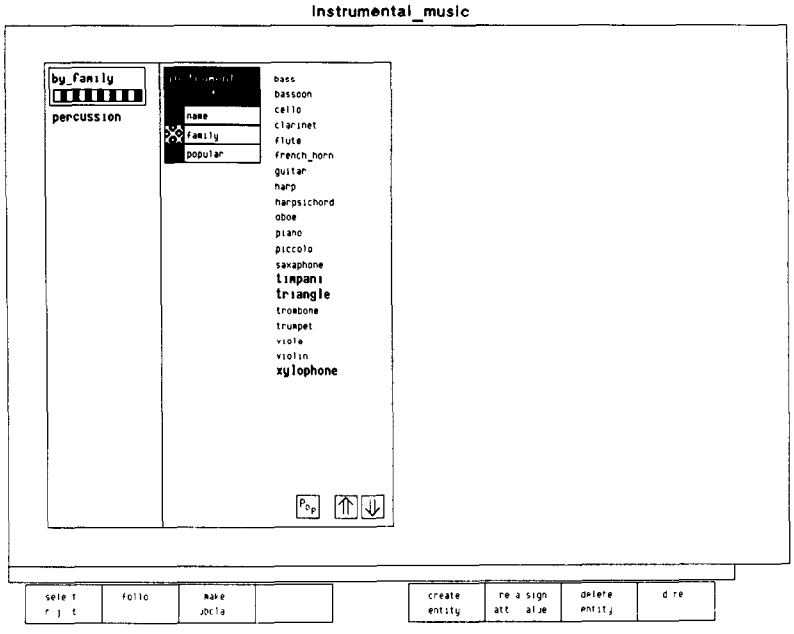

Figure 7. After following *percussion* (from the *by_family* grouping) into the *instruments* class

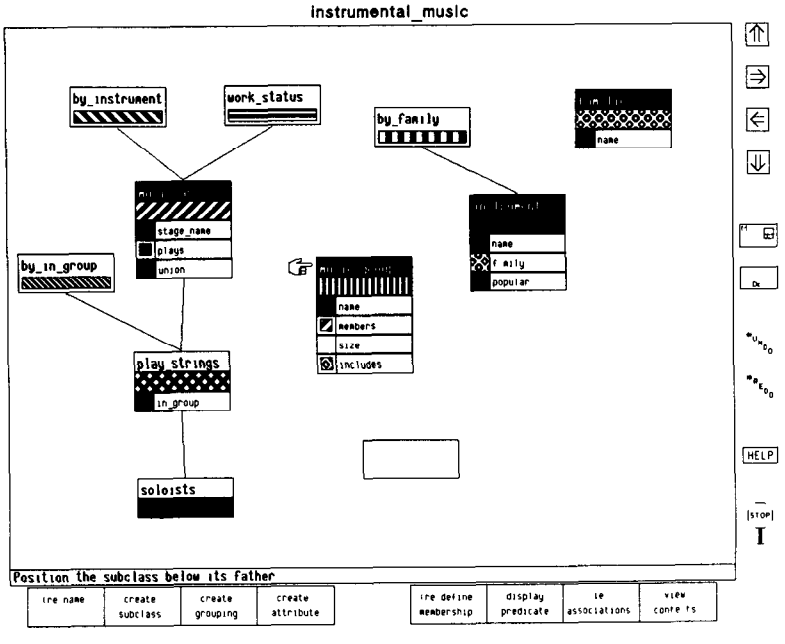

Figure 8. Creating a subclass of *music_groups*

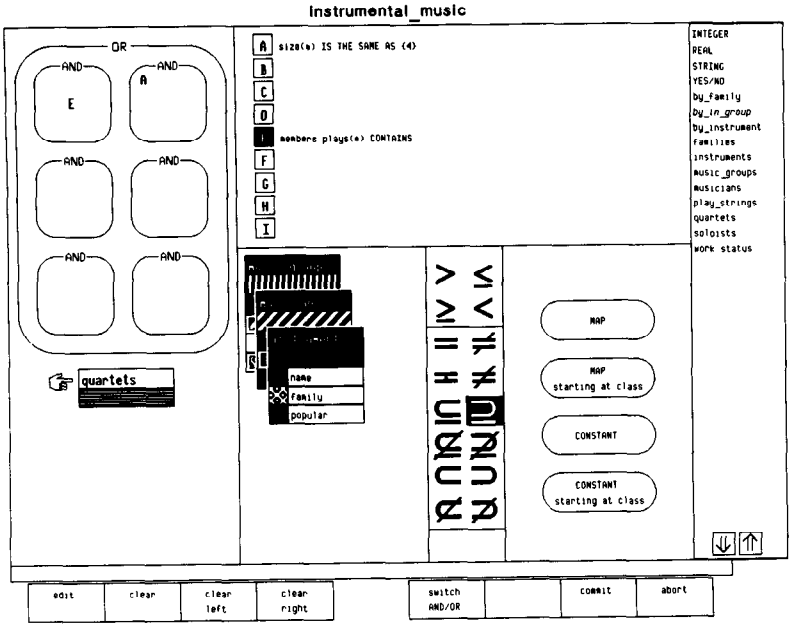

Figure 9. Constructing a predicate to define the membership of the *quartets* class

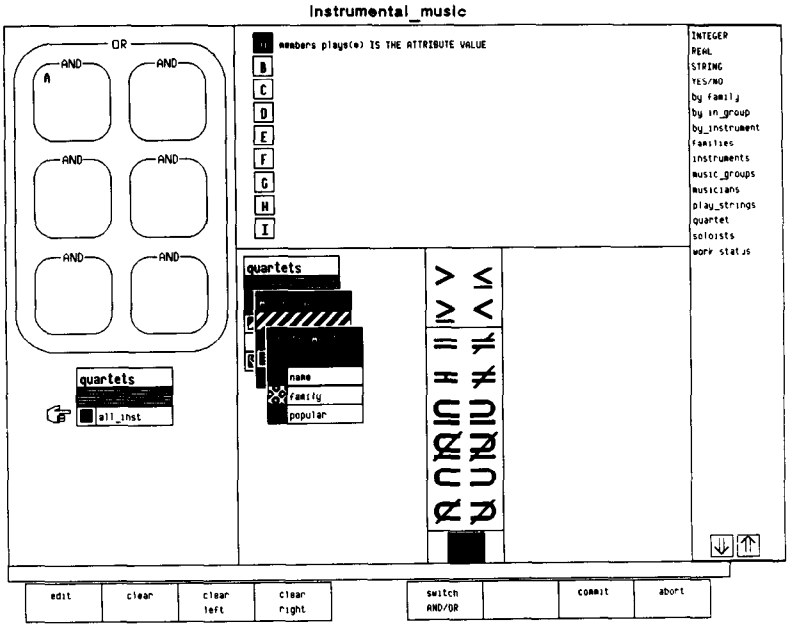

Figure 10. A completed derivation for the attribute *all_inst* in the *quartets* class

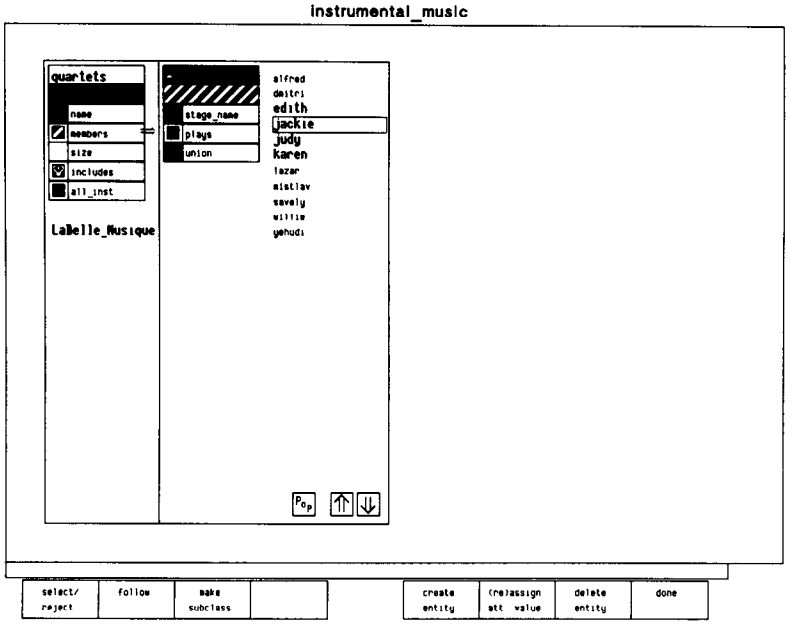

Figure 11. Changing the data selection.

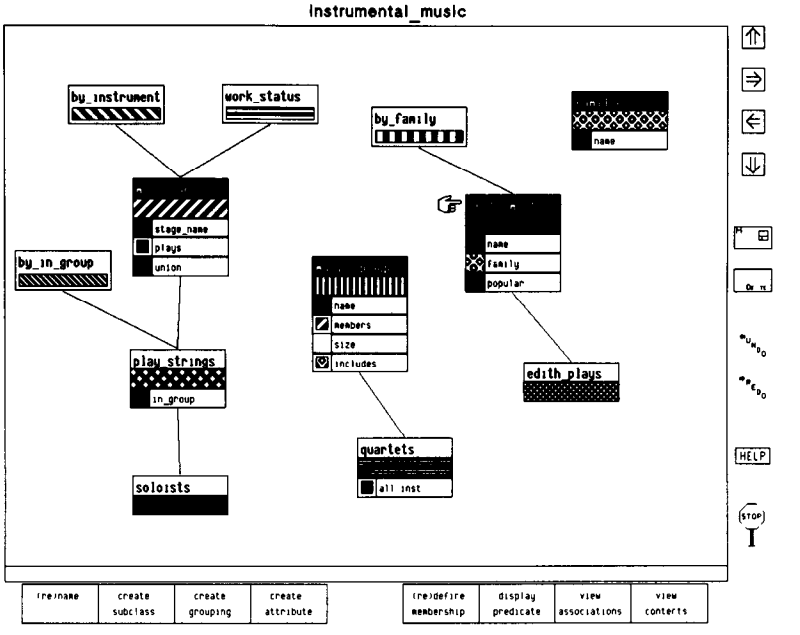

Figure 12. The inheritance forest with the new user-defined subclass *edith_plays* that was created at the data level

References

[BF] P Buneman, R E Frankel, "FQL – A Functional Query Language", In Proc ACM SIGMOD Int Conf Management of Data, Boston, Mass, 1979

[BN] H Biller, E J Neuhold, "Semantics of Databases The Semantics of Data Models", Inf Syst. 3 (1978), 11-30

[BS] M Brown and R Sedgewick, "A System for Algorithm Animation", Brown University, Department of Computer Science, Technical Report No. CS-84-01

[Ca] R G G Cattell, "An Entity-based Database User Interface", In Proc ACM SIGMOD Conf Management of Data, May, 1980

[Ch] P P S Chen, "The Entity-Relationship Model Towards a Unified View of Data", ACM TODS 1, 1, March 1976

[Co1] E F Codd, "A Relational Model for Large Shared Data Banks" Communications of the ACM 13, 6 (June 1970) 377-387

[Co2] E F Codd, "Extending the Database Relational Model to Capture More Meaning" ACM Transactions on Database Systems 4, 4 (December 1979), 397-434

[Fo] Fogg, D , "Lessons From "Living in a Database" Graphical Query", ACM SIGMOD, 14, 2, Proceedings of the Annual Meeting, June 18-21, 1984

[He] C F Herot, "Spatial Managment of Data", ACM Transactions on Database Systems, Vol 5, No 4, December 1980, pages 493-514

[HM] M Hammer, D McLeod, "Database Description with SDM A Semantic Database Model", ACM TODS 6, 3, September 1981, 351-387

[HY] R Hull, C K Yap, "The Format Model A Theory of Database Organization", JACM, Vol 31, No 3, July 1984, pages 518-537

[Ki] R King,"Sembase A Semantic DBMS",Proceedings of the First International Workshop on Expert Database Systems, Kiawah Island, South Carolina, October 1984

[MBW] J Mylopoulos, P A Bernstein, H K T Wong, "A Language Facility for Designing Database-Intensive Applications", ACM Transactions on Database Systems, Vol 5, No 2, June, 1980, pages 185-207

[MS] N McDonald and M R Stonebraker, "CUPID - The Friendly Query Language", Proceedings of the ACM Pacific Conference, San Francisco, April, 1975

[RBBCFKLR] D Reiner, M Brodie, G Brown, M Chilenskas, M Friedell, D Kramlich, J Lehman, and A. Rosenthal, "A Database Design and Evaluation Workbench Preliminary Report", Proceedings of the International Conference on Systems Development and Requirements Specification, Gothenburg, Sweden, August, 1984

[Re] S Reiss, "Graphical Program Development with PECAN Program Development System", Brown University, Department of Computer Science, Technical Report No CS-84-04.

[S] D W Shipman, "The Functional Data Model and the Data Language DAPLEX", ACM TODS 6, 1 (1981), 140-173

[SK] M Stonebraker, J Kalash, "TIMBER A Sophisticated Relational Browser ", Technical Report, University of California, Electronics Research Laboratory, May, 1982

[SS] J M Smith, D C P Smith, "Database Abstractions Aggregation", CACM 20, 6 (1977)

[TL] D C Tsichritzis, F H Lochovsky, "Data Models", Prentice-Hall, 1982

[WMy] H K T Wong, J Mylopoulos, "Two Views of Data Semantics A Survey of Data Models in Artificial Intelligence and Database Management", INFOR 15, 3 (1977), 344-382

[Zd] S B Zdonik, "Object Mangement System Concepts", Proceedings of the Second ACM-SIGOA Conference on Office Information Systems, Toronto, Canada, June, 1984

[Zl] M M Zloof, "Query by Example The Invocation and Definition of Tables and Forms", Proceedings of the First International Conference on Very Large Databases, September, 1975