Ski: A Semantics-Knowledgeable Interface

Roger King and Stephen Melville

University of Colorado
Department of Computer Science
Boulder, Colorado, 80309

Abstract

The Semantics-Knowledgeable Interface (Ski) is designed to provide a powerful - but usable - interface to databases. To do this, it combines the expressiveness of semantic models, the broad communications "bandwidth" of graphics, and a novel approach to the representation and manipulation of database schemas.

1. Introduction

This paper describes a database interface which is under construction. Ski, the Semantics-Knowledgeable Interface, is built on top of an existing semantic database implementation called the Semantic Database Constructor (see [FKM84]). Ski is intended as a naive interface, but allows access to a full semantic data definition and manipulation language called Semdal (see [KS84]). The implementation, language, and interface together form a semantic DBMS called Sembase.

Sembase is based on a semantic model. Semantic modeling in general is discussed in [KM84b]; briefly, semantic database models use a semantic network approach in providing data modeling constructs which are more expressive than traditional data models (i.e., relational, hierarchical, network). A semantic model provides facilities for defining non-atomic objects (abstraction) through attributes and for constructing type/subtype hierarchies (generalization). The model underlying Sembase is described in more detail in [KM82] and [KM84a].

The next section provides a brief overview of Ski. Section 3 describes the screen layout of Ski. Due to space limitations a simple example, rather than a complete system description, is given in Section 4.

This work was supported by IBM under a Faculty Development Award (1984).

2. Overview

A few researchers have examined the issue of using a semantic database as the framework for a graphics-based user interface (e.g., MPS [Wi83]; and GUIDE [WK82]). A (non-graphics) interface based on a semantic model is described in [Mo83]. In [SK82] a graphics-based relational browser is described; while it does not use a semantic model, it does support powerful window and icon manipulations.

Semantic models are both expressive and complex. The challenge in creating a semantic interface lies in making semantic modeling a manageable tool. Ski does not rely on a statically defined graphical layout of the schema. Instead, we attempt to use the semantic relationships inherent in the schema definition to dynamically drive the graphical representation. The portions of the database to be displayed and the types of relationships depicted vary with each user's area of interest, as indicated through their use of the SKI operators.

Ski uses a formated screen to simplify the representation of semantic schemas and supports semantically-motivated operators. While interacting with Ski, the user uses these operators to create a *session view*. The user selects schema components of interest and peruses the schema for related information. The unique feature of Ski is that this perusing is not performed navigationally, but semantically. The user may also reconfigure or access the database using the Ski operators.

Ski provides three classes of operators for perusing the schema while creating a session view. Ski allows the user to: (1) explore the direct semantic relationships in the schema by examining attributes, subtypes, constraints, and parent types, (2) determine how two schema components are related by locating and displaying the paths through the schema which connect them, (3) view the *update environment* of a specific schema component with respect to a particular update operation by displaying the schema components which are likely to be affected by performing that operation on the specified component.

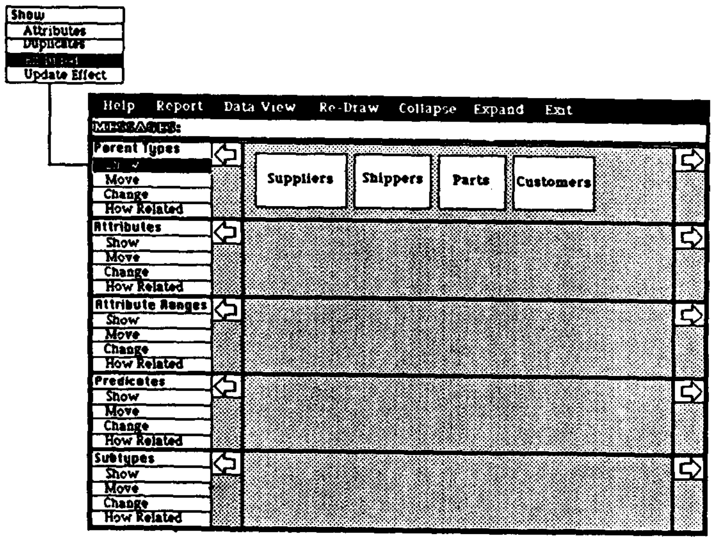
These last two sorts of perusing are actually quite complex, due to the existence of subtypes. The predicates which define the membership of subtypes may involve schema components which structurally are from very different parts of the schema. As an example, a smart student may be defined as one who gets an A in a course such that the professor who teaches that course is known to give tough grades. The information about a professor's grading history may not be "near" the student's information schematically, but the two are very closely related.

Ski consists of six window subsystems. First, the user enters the Hello subsystem, which provides a brief summary of the functions of Ski, points out the existence of the Help subsystem, and allows any previously saved session views to be recalled. The Schema View subsystem is then entered; it is used to construct and/or peruse a session view. The Data View subsystem is used to examine the attribute values and subtype memberships of the session view. The Data View also provides access to the language Semdal. The Report Generation subsystem is used to format data for output. The Help subsystem documents the structure of Ski and provides tutorials on the use of the various operators. Finally, the Exit subsystem is used to store a session view.

3. The Formated Screen

While constructing a session view using the Schema View subsystem of Ski, the user is presented with a formated screen (see figure 1). The screen provides a medium for representing the complicated structure of semantic schemas. The screen is broken into a variable number of horizontal stripes. The top stripe represents one or more parent types, which may be any of the types or subtypes in the schema. The next stripe represents attributes of the parent types.

Attribute ranges are shown in the third stripe. Predicates based on any of the parent types are shown in the next stripe and subtypes of the parent types are shown in the fifth stripe. Then, the sixth stripe (not shown in the figure) represents attributes of the range types from the third stripe. The seventh through ninth stripes show the ranges, predicates, and subtypes of the range types from stripe three. The stripes continue in this fashion. Any predicate stripe may reference a type or subtype from the parent stripe or any range stripe.



The user performs a Show and chooses Excluded Types from the submenu.

Figure 1.

A three button mouse is used to control Ski operations. The Drag button is used to scroll, expand, cóllapse, or move icons on the screen. The Select button is used to bind a schema component on the screen as a parameter to an operation. The Menu button is used to choose Ski operations. When a menu item (on the left border of figure 1) is picked with the Menu button, a general category of operations is chosen. Another menu (see the box at the top left of figure 1) then appears. One more pick with the Menu button chooses a specific operation.

Any component placed on the main screen becomes part of the session view. Certain operations (as described below) use a separate screen. In order for objects appearing on a separate screen to be placed in the session view, the Include operator must be used.

The user may scroll up or down to view all stripes; for added screen space, any stripe may be removed from the screen (but not from the session view) by selecting the Collapse operator from the top menu stripe (in the black border on figure 1) with the Menu button. Expand is used to make a stripe reappear. Also, square icons may appear at either end of each stripe (see figure 4). If there is a token which has "fallen off" a stripe, a square will appear; more than one square means that a number of tokens have fallen off. The user may scroll to see the end of the stripe to find these tokens. Scrolling is performed by picking the appropriate arrow (above the squares) with the Drag button of the mouse.

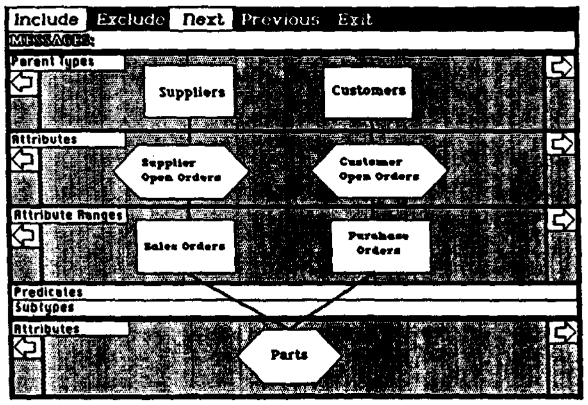
4. A Sample Session

The ABC Widget Company is a distributor of widgets. Customers submit sales orders for desired parts. If there are not enough parts in inventory to fill a sales order, a purchase order is placed with a supplier. Unfortunately, ABC is suffering from a cash flow problem. They have no money to pay suppliers. Mr. Fred Jones, the president of ABC, is about to use the Ski user interface to assess the consequences of this problem.

Mr. Jones begins his quest by entering the Schema View subsystem and using the menu button to choose the operator Show under the Parent Types stripe (see Figure 1). He selects Show Excluded from the submenu. This chooses all root types (not subtypes) which are not currently in the session view and adds them to the view.

Mr. Jones is trying to find out what will happen to his customers if he does not pay his suppliers. First, he selects Suppliers and Customers (with the Select Button). The selected components are displayed in reverse video to provide visual confirmation that the desired types have been selected. He then uses the Menu button to invoke the How Related operator with respect to the selected types.

The How Related operator gives Mr. Jones a separate screen showing the shortest path through the schema connecting Suppliers and Customers (see figure 2).



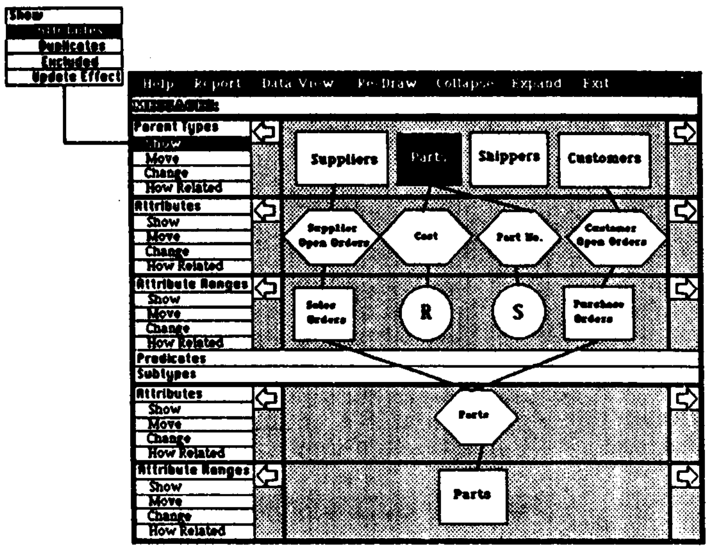
The user requests the Next shortest How Related path. The results are to be included in the session view.

Figure 2.

Invoking the Next operation on the How Related submenu would cause the next shortest schema path to be displayed. Mr. Jones "picks" the Include operation (using the Menu button on the mouse) to add the displayed components to his session view, and exit the How Related screen.

Note that on the How Related screen, only those schema components lying on the connecting path are displayed. If Ski relied upon a pre-defined (i.e. static) graphical representation of the schema, any attempt to show how two schema components were related would be likely to include considerable "noise"; that is, additional schema components not relevant to the current query. Using a dynamic representation allows rapid isolation of pertinent schema components.

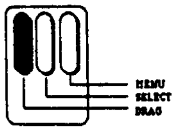
Mr. Jones would like some more information about Parts. He is searching for some reference to money. He uses the Select button to choose Parts and then chooses the Show Attributes operation on the Parent Types stripe. The result is shown in figure 3. The four atomic types (Real, String, Boolean, and Integer) are represented by circles. So, in figure 3, we see that Parts have an attribute "Cost" whose range is Reals and an attribute "Part No." whose range is Strings. (Note that Mr. Jones has also collapsed the Predicate and Subtype stripe in order to simplify his screen image. This was performed by picking the stripes to collapse with the Select button and then choosing the Collapse operator from the top of the screen with the Menu botton.)

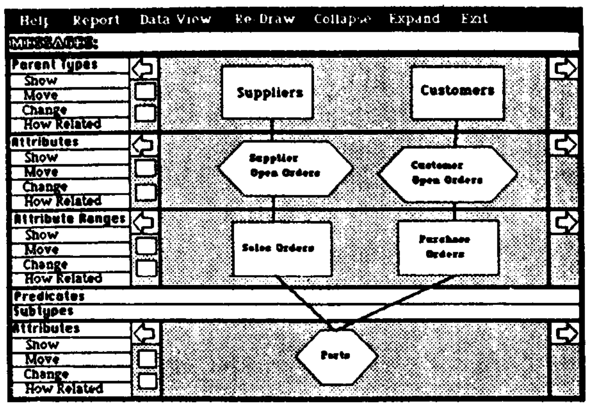


The user performs a Show Attributes operation on the selected Parent Types.

Figure 3.

On the Parent Types Move submenu (not shown) is a Move Bundle operation. This allows selected Parent Types (together with their attributes, and Subtypes) to be horizontally dragged *en masse* to a less cluttered portion of the session view. Mr. Jones selects Suppliers and Customers, invokes the Move Bundle operation, and using the Drag button, moves the bundle to the right of the screen. The result is shown in figure 4. The continuation icons have appeared, informing Mr. Jones that there is information off screen. It should be noted that a



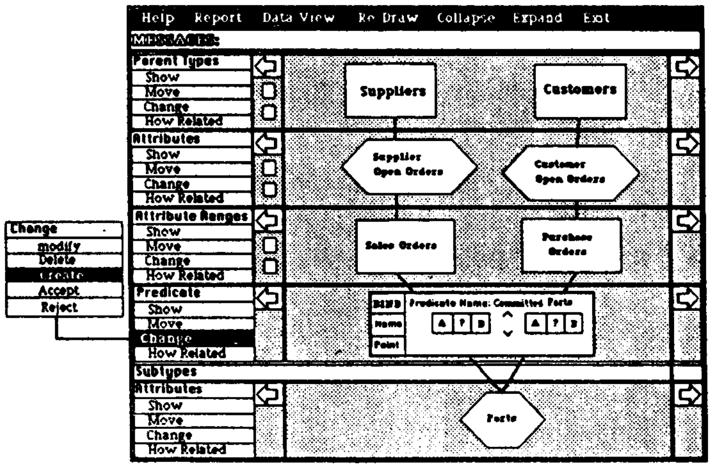


Using the mouse, the Bundle is dragged to the right of the screen.

Figure 4.

Drag operation may result in a number of crossed lines; the user may use the Re-Draw operation from the top border to simplify line positions.

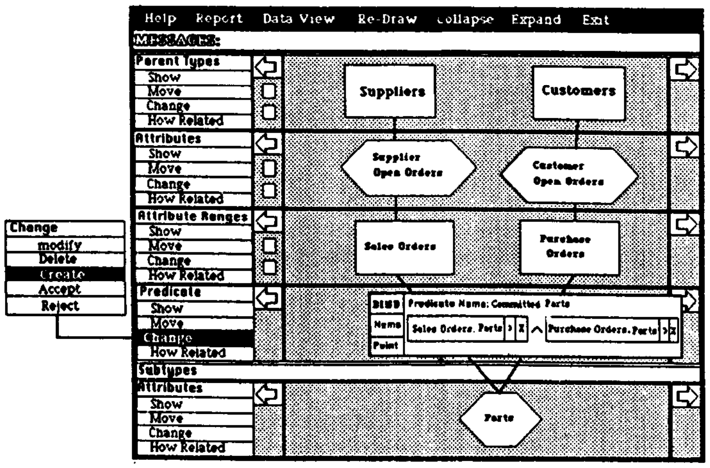
It is time for Mr. Jones to isolate the data he needs. He expands the Predicate stripe, and chooses its Change operation. This will allow him to alter the schema. After choosing the Create operation, Mr. Jones uses a skeleton (figure 5) to create his predicate. What he wants is all parts that are on both sales and purchase orders. He may bind the attributes in the predicate with either the Select button or by using text.



The predicate of the Subtype is specified.

Figure 5.

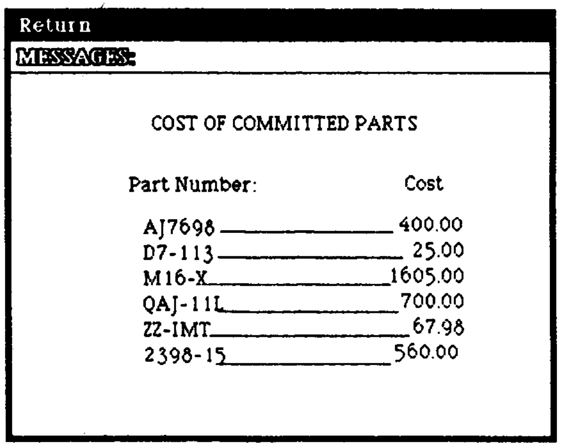
In the example, Mr. Jones writes in Parts of Sales Orders and Parts of Purchase Orders (using dot notation), two set inclusion operators (chosen from a menu), the free variable X, and the logical AND operator (which may also be chosen from a menu). He names the new subtype Committed Parts (see figure 6).



The predicate of the Subtype is specified.

Figure 6.

Mr. Jones' last step is to enter the Data View subsystem to view the Cost attribute of each part in Committed Parts (see figure 7). This lets him know how much capital he must find to satisfy his customers. He may want to print this out using the Report subsystem.


Return

MESSAGES:

COST OF COMMITTED PARTS

Part Number:	Cost
AJ7698	400.00
D7-113	25.00
M16-X	1605.00
QAJ-11L	700.00
ZZ-IMT	67.98
2398-15	560.00

The Corresponding data is viewed.

Figure 7.

5. System Status

Ski is currently being constructed to run under UNIX on high-resolution graphics workstations (SUNs). The underlying semantic database implementation (Sedaco) and the language Semdal have been fully implemented and are operational on the SUNs.

Acknowledgements

The authors would like to give their sincere thanks to Vickie Clark and Scott Hudson who produced the figures for this paper.

References

* [FKM84]

  Farmer, D., R. King, and D. Myers, "A Tool for the Implementation of Databases", in *Proceedings of COMDEC '84, Computer Data Engineering Conference*, Los Angeles, CA, April 1984.

* [KM82]

  King, R. and D. McLeod, "The Event Database Specification Model", *Proceedings of the Second International Conference on Databases: Improving Usability and Responsiveness*, Jerusalem, Israel, June 1982.

* [KM84a]

  King, R. and D. McLeod, "An Approach to Database Design and Evolution", in *Conceptual Modeling*, editors M. Brodie, J. Mylopoulos, and J. Schmidt, 1984.

* [KM84b]

  King, R., and D. McLeod, "Semantic Database Models", in *Database Design*, editor S. B. Yao, 1984 (to appear).

* [KS84]

  King, R. and S. Sanke, "A Semantic Data Language", *Proceedings of the 1984 Trends and Applications Conference*, May 23-24, 1984.

* [Mo83]

  Morgenstern, Matthew, "Active Databases as a Paradigm for Enhanced Computing Environments", *Proceedings of the Ninth International Conference on Very Large Databases*, October 31 - November 2, 1983 pages 34-42.

* [SK82]

  Stonebraker, M. and J. Kalash, "TIMBER: A Sophisticated Relational Browser", *Proceedings of the Eighth International Conference on Very Large Databases*, 1982 pages 1-10.

* [Wi83]

  Wilson, G., E. Domeshek, E. Drascher, and J. Dean, "The Multipurpose Presentation System", *Proceedings of the Ninth International Conference on Very Large Databases*, October 31 - November 2, 1983 pages 56-69.

* [WK82]

  Wong, W.K.T., and I. Kuo, "Guide: Graphical User Interface for Database Exploration" *Proceedings of the Eighth International Conference on Very Large Databases*, Mexico City, 1982, pages 22-32.