FaceKit: A Database Interface Design Toolkit

Roger King
Michael Novak

Department of Computer Science
University of Colorado
Boulder, Colorado 80309

Abstract

FaceKit is an interface design toolkit for object-oriented databases. By combining techniques from the realm of User Interface Management Systems (UIMS) with a built-in knowledge about the specific kinds of techniques used by object-oriented databases, we have designed a system that allows a user to customize a database interface with minimal programming. Knowledge about object-oriented database constructs such as classes, groupings, etc., allows FaceKit to semi-automatically create graphical constructs appropriate to the object-oriented database environment. FaceKit is built on top of Cactis, an object-oriented database management system (DBMS), and is capable of creating interfaces that deal with Cactis both at the schema and data level.

Keywords: graphical interfaces, user interface management systems, object-oriented databases.

1. Introduction

FaceKit is a window based, interactive graphical system for designing (we do not support a formal design phase; some people might call FaceKit a tool for *building* an interface) graphics-based interfaces for object-oriented databases. Although more of a UIMS than a simple database interface, FaceKit is intended for designing a particular set of interfaces - those dealing with object-oriented databases. FaceKit knows about schemas, type-subtype hierarchies, methods, and database tools such as data definition languages (DDL). Therefore, knowledge about the types of interfaces being designed and the objects they manipulate is built into FaceKit. This knowledge allows default representations of database objects and representations defined or modified by the user. Both default and user-defined representations inherit properties in the same way as the database objects they represent. This allows the user to easily change any class or subclass of object representations and also leave the defaults in effect where desired. By taking advantage of the structure provided by the database, FaceKit makes designing a specific interface easier and faster, and forces the interface to stay consistent with the database.

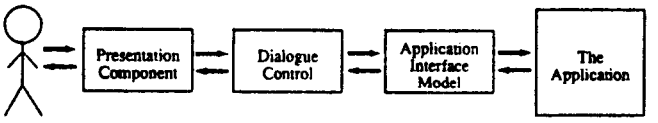

Figure 1

Database interfaces have undergone many changes in the last few years. Until a few years ago, most of the database interfaces we saw where non-graphical. Aside from programming language interfaces, they include relational algebras such as ISBL [Ull], relational calculi like QUEL [HSW75], and query languages such as SEQUEL [BoC74], which falls somewhere in between the two. These interfaces vary in many ways, but their general goal is to provide a general, fairly complete, interface to a database. Then came interfaces such as QBE [Zlo75], which provides a form style interface for specifying queries. In the last few years there has also been much interest in graphical database interfaces. Among these are interfaces such as ISIS [GGK85], Ski [KiM84], and SNAP [BrH86], which allow schema manipulation in an interactive graphical environment. They also include office forms systems such as FORMANAGER [YHS84], Freeform [KiN87], and SPECDOQ [KGM84]. These systems provide a non-expert interface for the storage and retrieval of office data.

Obviously, a single interface cannot be everything to everyone. Since many different database interfaces may prove useful, we feel that being able to quickly design a new or altered interface could prove very useful. UIMS's are designed for this very purpose. The question is, why not use an existing UIMS to rapidly generate interfaces for an object-oriented database? To answer this question, a little background on UIMS's is necessary.

We will use the Seeheim model of user interfaces [Gre84, Gre85] (figure 1) to examine and compare some of the existing UIMS systems. The Seeheim model is a logical model applicable to a wide variety of UIMS's. In this model the user interface is broken into three logical components: the presentation component, the application

This work was supported in part by ONR under contract numbers N00014-86-K-0054 and N0014-88-K-0559.

interface model, and the dialogue control. The presentation component is responsible for producing device output and gathering user input. The application interface model is responsible for representing application data and making it available to the interface, as well as providing the application with access to the interface. The dialogue component forms the bridge between the presentation component and the application interface model. It makes sure that the application carries out user requests and that the presentation component produces the output requested by the application.

The main emphasis in much of the UIMS research has been on the dialogue control component [Gre86]. Although there have been systems with dialogue models based on transition networks, grammers, and events, these system share a common perspective. UIMS's such as ADM [SRH85], Grins [ODR85], Menulay [BLS83], MIKE [Ols86], and Trillium [Hen86] all view an interface as a dialogue between the user and the application. The user makes a request and the application responds by performing some task and possibly providing some output. Similarly, the application makes input requests which the user responds to. The research emphasis in these systems has been on how to provide the user with the appropriate tools for specifying the dialogue.

Work done on the presentation component includes Peridot [Mye87] and Squeak [Car87] Both of these systems put heavy emphasis on allowing users to create new interaction techniques. Peridot users create these techniques by using examples to show how they should appear, while Squeak lets users apply direct manipulation principles [HHN, LeL83] to specify them.

There are also several systems which focus on the application interface model. Filters [Ege88] and Coral [SzM88] each provide a method of specifying relationships between application and interface objects. GWUIMS [SHB86] and Higgens [HuK88] both allow for sharing of data between the application and the interface. All of these systems use an object-oriented data model.

Another object-oriented system called GROW [Bar86] looks at some dialogue and presentation issues. GROW emphasizes building modifiable and reusable interfaces. The dialogue component uses messages for communication between the application and the interface. The presentation component provides a kernel of graphical objects arranged in a taxonomic hierarchy and it allows the user to specify inter-object relationships.

There are two main reasons why we don't wish to use any of the currently available UIMS's. First, even though we have seen systems which can communicate with application data in some manner, a general purpose UIMS has no knowledge of database schemas. Since FaceKit knows about database schemas, users may access database objects, methods, etc. and incorporate them directly into the interface. Similarly, interface objects are actually stored in the database and have the same structure as database objects. Thus, only one view of objects exists. Even those UIMS's which share some data between the interface and the application still require the user to view interface and application objects separately and to provide information about how the two are related. Having only one view of objects means that less information needs to be provided by the user. Furthermore, users do not need to keep track of two different types of objects.

By providing access to database tools such as query languages and methods, FaceKit also gives the user more ways of rapidly constructing an interface. Instead of writing code to generate an interface technique, the user may invoke a method in the database, or use a query language to define the technique. Interface objects previously created may also be used, because they too are stored in the database.

The second reason that we don't wish to use a currently available UIMS is that creating a new database interface often involves creating a new application. For example, an office forms interface involves much more than merely an interface to the database. New functionalities must be built into the interface. These may include new mathematical functions such as computing the city and state sales tax on a sales field, and utilities such as an inter-office memo system. Both of these new functionalities involve more than simple interaction with the DBMS. In order to support this type of interface, we wish to allow the user to interactively design both an interface and its corresponding application simultaneously [HuK88]. Unlike in a UIMS, our approach treats the interface being designed as a database environment, rather than a dialogue between a distinct user interface and an application. Instead of defining merely an interface, we define the visual, functional, and interactive aspects of the environment. Thus, certain otherwise hard to obtain functionalities such as binding representations to objects can be achieved. This also allows for "realistic" default interfaces and faster specification of representations. In fact, such an integration of database and UIMS technology has been suggested before [Gre87, Ols87].

The minus side of our approach is that neither interfaces outside the object-oriented database realm nor non-Cactis database applications are supported. However, many applications may be be placed into an object-oriented database. For example, much of the circuit board design software currently available uses simple files to store circuit information. Such systems could easily fit into the object-oriented database paradigm. One could argue that even aside from making it possible to use FaceKit with the application, storing the data in a database would make the application itself more manageable.

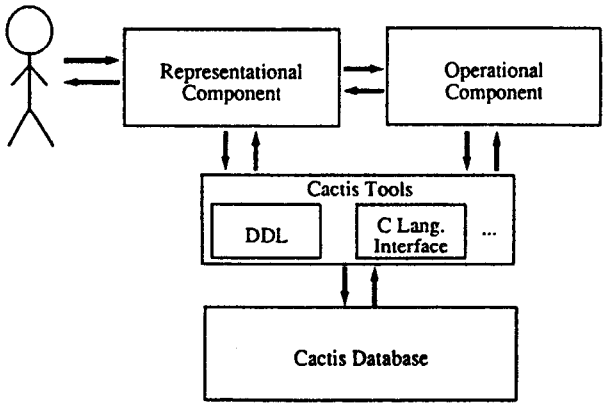

Figure 2

2. Modeling an Interface

FaceKit is built on top of an object-oriented DBMS named Cactis [HuK86, HuK87]. Since FaceKit has knowledge about both Cactis schemas constructs and the various schema manipulation tools (DDL, C language interface, etc.) available within Cactis, we will give a brief description of the Cactis data model before proceeding to describe the data modeling aspects of FaceKit.

Cactis views an application environment as a collection of *constructed objects*. An object may have *attributes* and *relationships*. Objects and relationships are typed. A constructed object's type is determined by two things: its attribute structure and its *connectors*. A connector allows a relationship to be applied to a certain object. An attribute is an atomic property of a constructed object. These atomic properties may be of any C data type, except pointer. A relationship is a directional mapping from one constructed object to one or more constructed objects. Restrictions such as non-null or unique may also be put on a relationship.

For example, a constructed object called person may have the attributes name, social_security_number, and age, which are all atomic and single-valued. It may also have a connector called children and a connector called parent. The directed relationships my_children and my_parent can be used to connect people to their immediate family. Relationships may be used to pass attributes from one object to another, in order to calculate derived attributes. In this way, the social security number of a person could be passed to a child over the my_children relationship, and used as the value of an attribute called my_parent's_social_security_number.

FaceKit uses the data model and the database management tools provided by Cactis for data and schema manipulation. An interface needs to communicate with these Cactis tools and it needs to manage visual representations and coordinate interface tasks. To accomplish this a FaceKit interface consists of two conceptual components: the *representational* component and the *operational* component. Both the representational and operational components communicate directly with Cactis (figure 2), but their tasks are quite different.

The primary responsibility of the representational component is managing the visual representations of database objects. For the purpose of this discussion, database object, or object, may refer to a constructed object, relationship, or attribute. Also, no distinction is made between schema and data objects at this time. The representational component builds, maintains and invokes the methods used to produce the visual representations of objects. Thus, if we build an interface where a data object called person is represented by a screen image of that person, the representational component makes sure that the method used to draw these screen images is invoked at the proper times, with the correct parameters.

The representational component is also responsible for the input and output associated with an interface. Input will generally consist of reading in some user data or command, while output will consist of either displaying new objects on the screen or invoking a different representation of objects already present. An object may have many different visual representations in the same interface. For example, a database may have a baby picture and an adult picture of the person in the example above.

The operational component is responsible for processing user queries and sending the results to the representational component so that the correct screen updates will be performed. The operational component performs a function much like that of the dialogue control in the Seeheim model. However, it has access to the Cactis database management tools. This means that no application interface model is necessary. Instead, the operational component talks directly to the application. This has several advantages. First, it allows FaceKit to store the operational description of an interface within Cactis. Therefore, Cactis manages both the data and the operational description. Not only does this make storage of interface descriptions convenient, but it also allows operations to be functionally dependent on anything present in the database. This allows an interface to change its behavior as the database is modified. Second, since these tools give the interface direct access to the application data, semantic feedback can be gathered merely by examining the relevant data. Similarly, constraints can be checked and adhered to.

Perhaps the most important effect of allowing the operational component access to these tools is utilized when designing interfaces. Instead of only being able to bind a user request to some application subroutine, the interface designer may construct a query using one of these tools. Since new interface functionality can be added quite rapidly in most cases, most of the functionality of Cactis can easily be plugged into an interface.

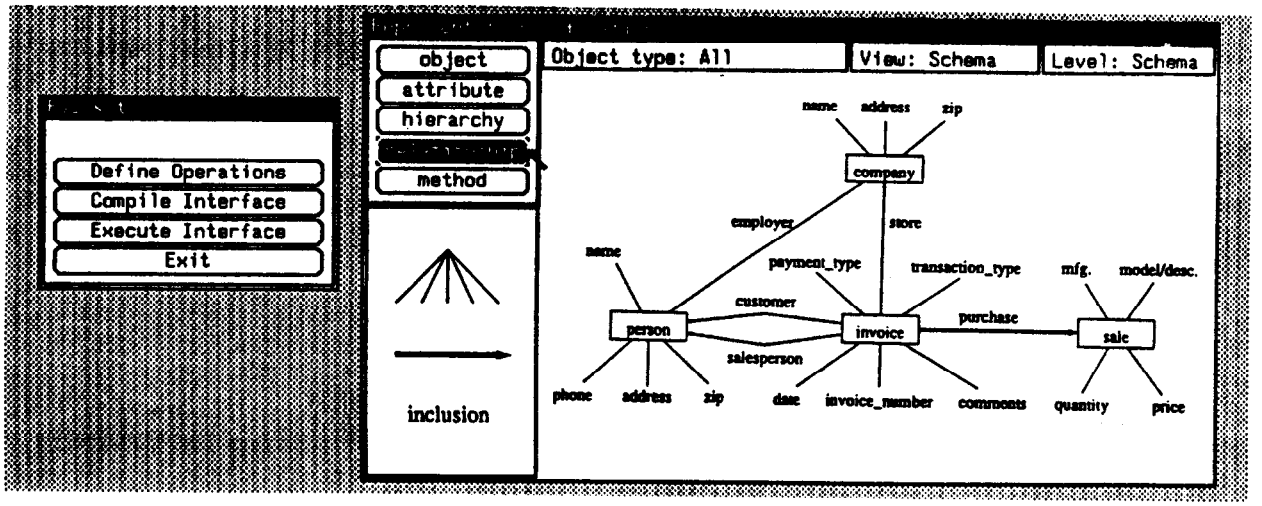

Figure 3

3. Designing an Interface

A very common UIMS approach to designing interfaces is to specify screen layout, then bind each possible user action to a specific application subroutine. Often, interface routines for the application program to call are also provided. FaceKit takes a different approach to designing interfaces. Our approach allows the user to define what we view as two somewhat different aspects of the interface: appearance, and functionality. When defining appearance we are really defining two kinds of visuals, interface constructs and database objects. By interface constructs we mean items such as menus, scrollbars, etc., as well as concerns like screen brightness, icon sizes, etc. Defining the appearance of database objects involves specifying representations for a class of objects. A representation may be identical for each data object in a class or it may be data dependent. In fact, it may be dependent on external data. For example, we may use the system clock to determine the brightness of a picture that represents the data object **sun**.

By defining the appearance of database objects separately, we need not worry about them when defining functionality. The type of the query result determines the screen appearance. Any type that has no user-defined representation will use a built-in default representation. For example, if a query results in an object of class **person**, the interface automatically uses the representation of person that has been defined. If one wishes to leave the screen alone and produce the query result elsewhere, this may also be specified.

Both appearance and functionality are defined interactively with FaceKit. In order to better explain how they are defined, we will show how one would go about defining an existing interface, an office forms system called Freeform [KiN87]. We will also show some sample applications of this interface. We will also show how an interface used to retrieve information about computer networks can be built using FaceKit.

3.1. Defining Representations with FaceKit

In figure 3, an object representation is being defined. The buttons on the left specify what kind of object is being defined and the line on top is a status line. In this case, the object type to be defined is **all** (the representation being defined is applicable to all types in the database). The current level and view are both **schema**. Two views are possible: **schema** and **class/subclass hierarchy**. The schema view is the standard graphical view of a database schema, while the hierarchy shows a forest of classes and subclasses. Two levels are also available: **schema** and **data**. The schema level lets us work with object types, while the data level lets us work with object instances. The commands for changing the view, level, and object type are available through a popup menu. The popup menu also has commands for placing restrictions on the representation being defined, moving around the schema, etc.

In the schema shown, the text enclosed in rectangles represents constructed objects and the unenclosed text represents the attributes of the constructed objects. The narrow arrows represent one-to-one relationships and the wide arrows represent one-to-many relationships.

Instead of representing a schema as a connected graph, we will represent relationships with inclusion. Thus, the object pointed to by an arrow will be included inside the object the arrow originates from. Also, attributes will be included within the constructed object they belong to. Once we choose **relationship** and **inclusion** (figure 3), the schema changes to look like figure 4. The label on top of each box is the name of the relationship

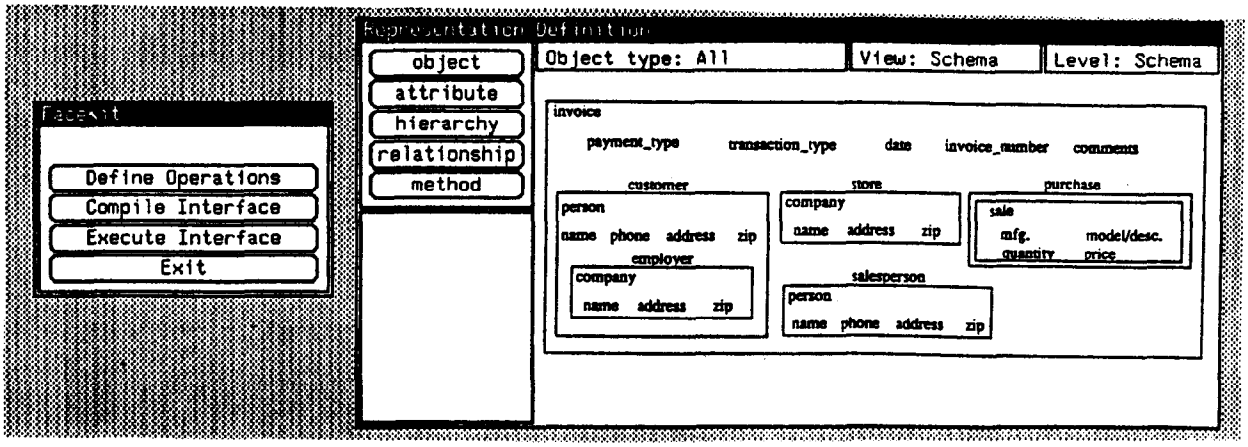

Figure 4

between the object represented by the box and the object it is included in. The double box around purchase denotes that it is a one-to-many relationship.

Although the representation looks fairly formlike now, some further refinement is still needed. Attributes need to have a line for filling in data behind them, representations need to be made different for various types of attributes, and multi-valued relationships such as purchase need to be represented with some appropriate structure like a table. Since the details of these changes are very similar to the examples above, we will not show them. Instead, we will give some examples of defining functionality.

3.2. Defining Operations with FaceKit

An operation definition window is shown in figure 5. The schema shown is a refined version of the schema in figure 4. Although there is no reason we cannot have both the representation definition and operation definition windows up simultaneously, we will only use one at a time in order to make our diagrams less crowded and confusing. A user building an actual interface would probably use them simultaneously.

By selecting an action (or sequence of actions) from the Action buttons and a result (or sequence of results) from the Result buttons, the user can bind the desired functionality to a set of actions. For example, suppose the sequence **pick**, **new action**, and **menu** is selected. This specifies the following sequence. The user picks an object.

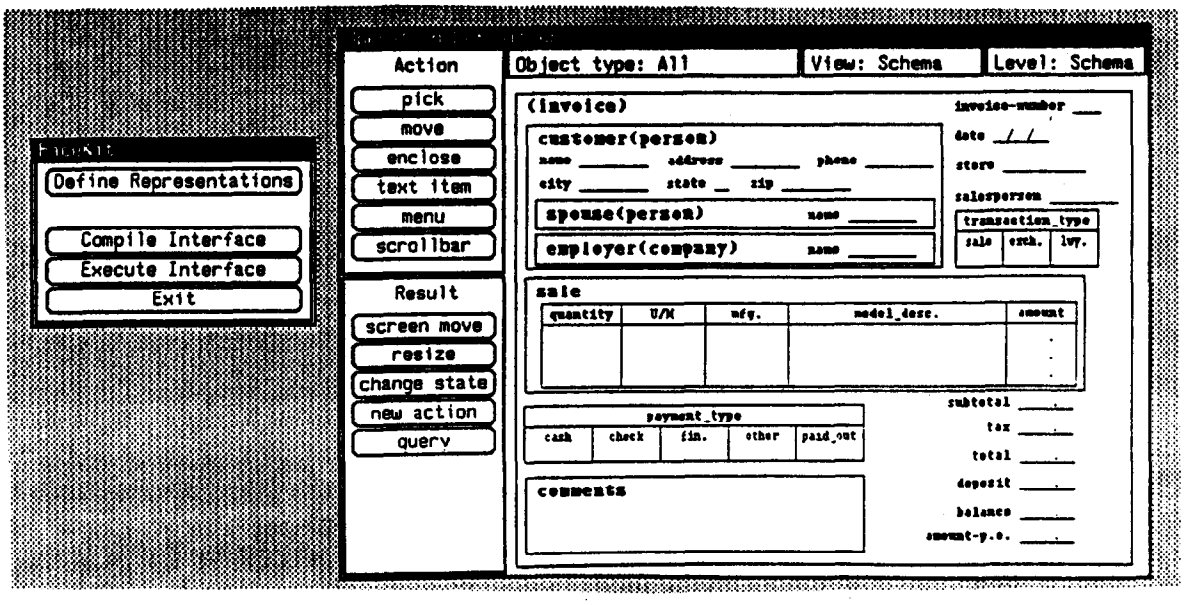

Figure 5

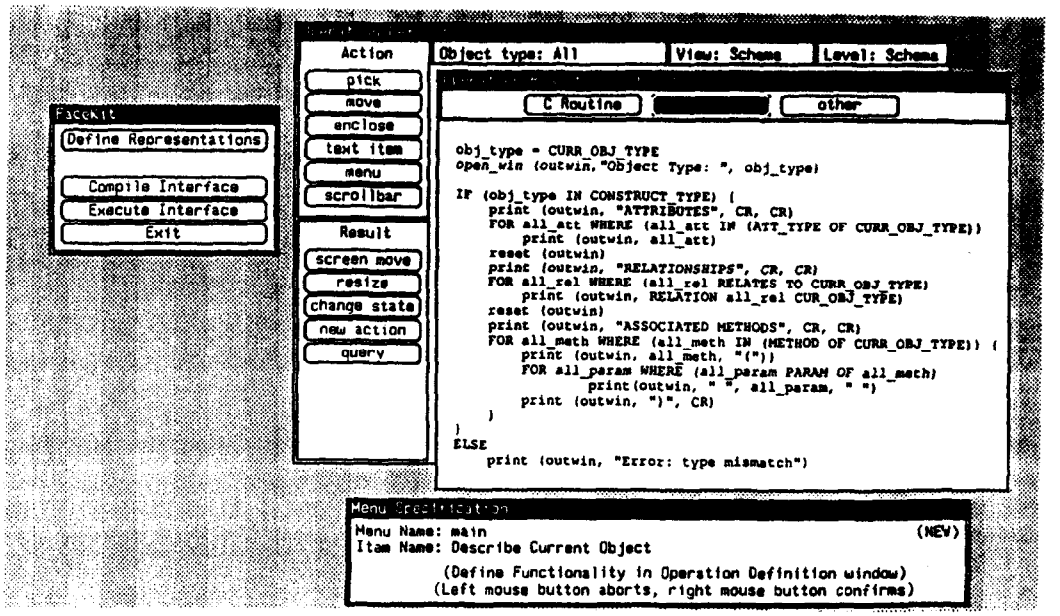

Figure 6

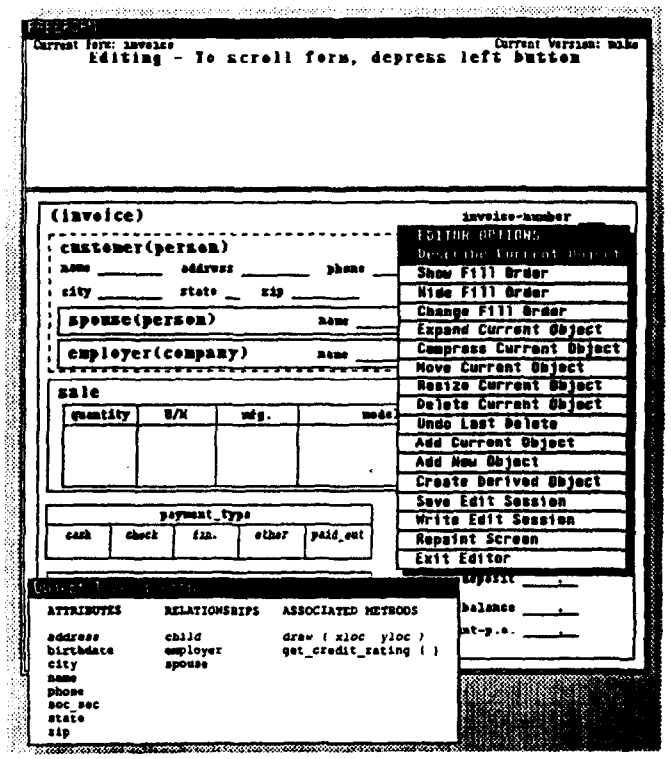

Figure 7

The result of this is a new action. This new action will be the appearance of a menu. Therefore, we are causing the a pick, in Freeform, to create a menu. Since a menu is to be created, a Menu Specification window is created, so that we may specify which menu to use. Figure 6 shows such a window, with a new menu being defined. We could have also chosen an already defined menu, but instead, we will demonstrate how one goes about defining a menu. This is done by defining each menu item and its corresponding functionality. The menu item being defined in figure 6 is **Describe Current Object**. The result of choosing this menu item will be a query, which will be defined in the Operation Result Specification window.

We will define this query using the DDL. The DDL being used is a simplified version of the Cactis DDL [SwS88], with some pseudo-code added. Open_win, print, and all variables in lower case are FaceKit constructs. All upper case words are DDL keywords.

First, the query opens an output window, then checks to make sure that the current object is of the class constructed. If it is not, an error message is displayed in the output window and the query is complete. Otherwise, all attributes, relationships, and methods directly connected to the current object's type are found. This information is formatted and displayed in the output window. The result of selecting the **Describe Current Object** menu item from within Freeform is shown in figure 7. The current object is the one surrounded by a dashed border. Both objects that appear on the form and those that do not, are found by the query. For example, the attribute birthdate is presented because it is in the database, although it is not part of the form being displayed.

Queries may also be produced by calling Cactis C routines. Currently, these are the only two query methods implemented. We will not give any examples of C queries, since they are conceptually similar to DDL queries.

3.3. A Software Engineering Example

Since most software engineers now have access to one or more computer networks, they need an easy way to

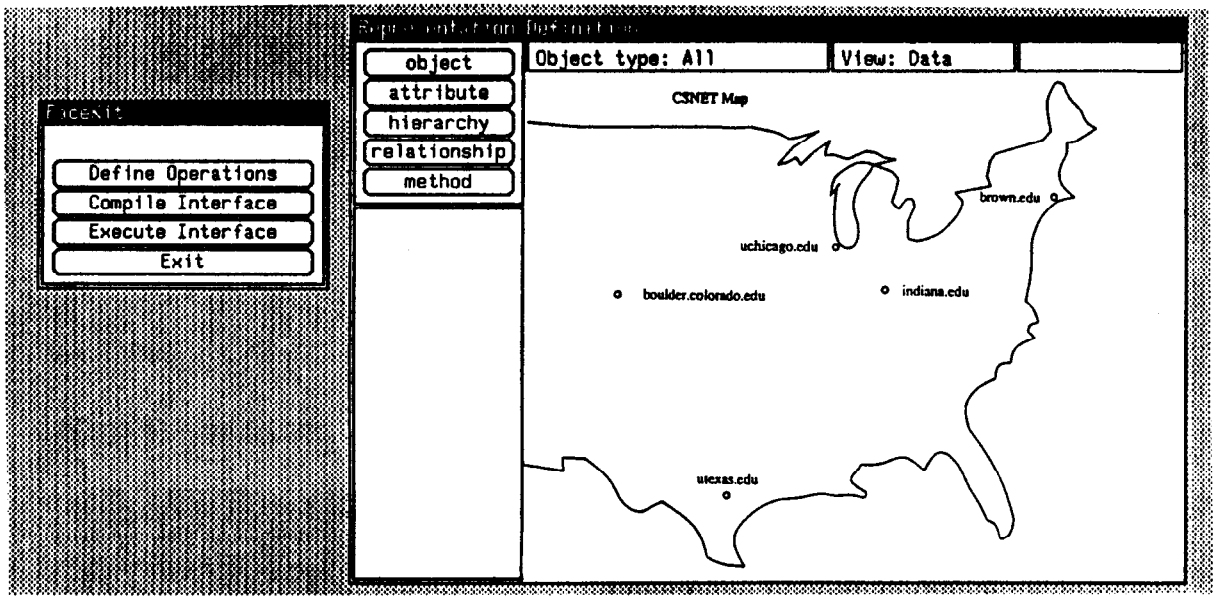

Figure 8

get information about these networks. With network configurations changing fairly often, storing the information in a database makes good sense. Having a graphical interface to access such information also makes good sense. Our next example shows part of such an interface being built with FaceKit. Since this interface does not look particularly "formlike", it is good for illustrating the variety of interfaces that can be built with FaceKit.

In Figure 8, we show a pictorial representation of a small subset of the CSNET sites in the United States. This representation was built by invoking a method (not shown) that plots database objects of type **CSNET_node**. Due to space constraints, only a small part of the map is shown on the screen. Note that actual data, not schema information, is being displayed.

Each **CSNET_node** contains a number of computers. Figure 9 shows a class hierarchy for the schema type **computer**. In this hierarchy, **computer** is the base type, **mainframe**, **workstation**, and **micro** are subtypes, and so on. Classes are represented by enclosed text and attributes are represented by unenclosed text. Attributes are connected with lines; subclasses are connected with arrows. Although this schema is not complete (for readability), we can see that computers such as a **Pyramid** and a **Sun 4** are different and contain different attributes. The database also contains methods which deal with this information. These methods are also accessible to FaceKit.

For example, a method which draws the representation of a computer may exist in the database (or one may be created specifically for this interface). FaceKit can access this information in order to create a graphical

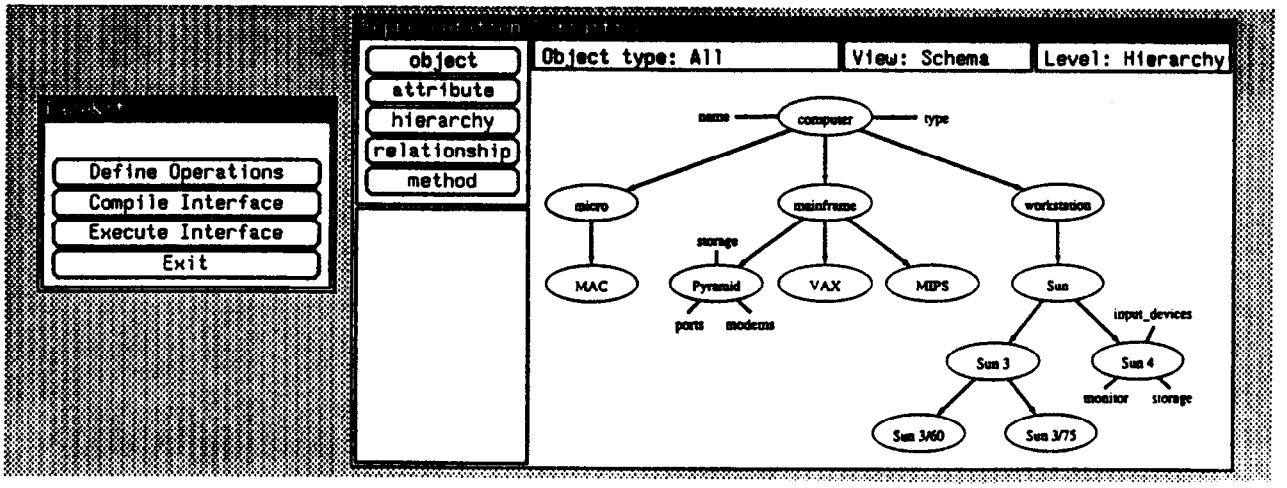

Figure 9

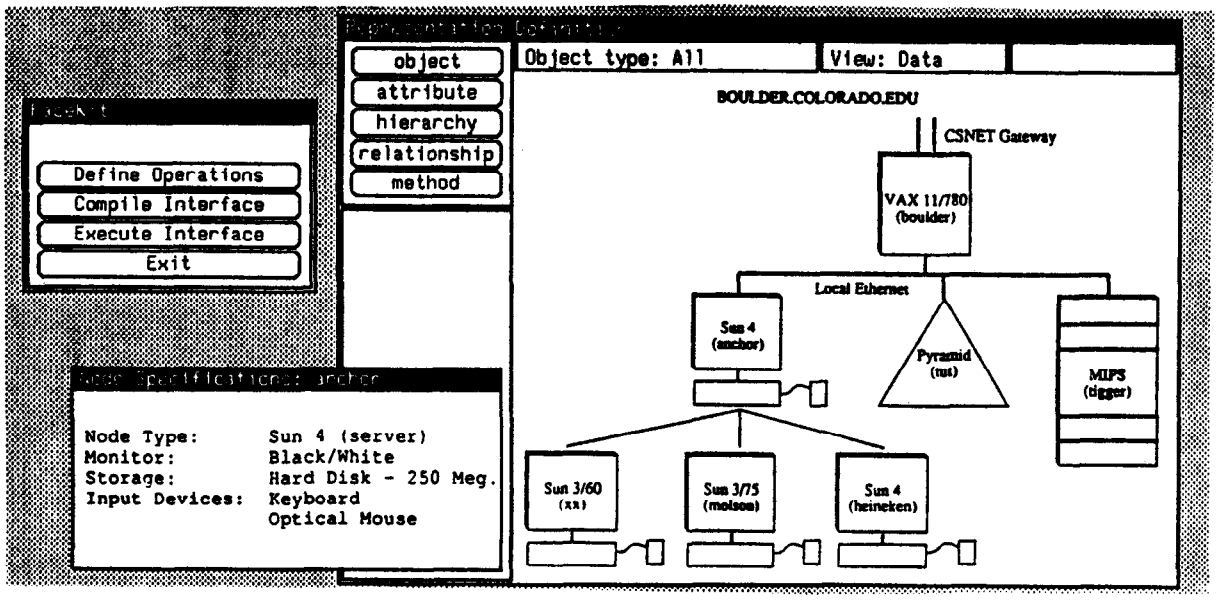

Figure 10

representation of a CSNET site (figure 10). Each computer at the selected site is shown along with its type and name. How they are connected to each other is also shown. In this example, the gateway to the CSNET is a VAX 11/780 named boulder. FaceKit does not even need to know the specifics about each computer at that site. For each computer at that site, the database knows its type and the proper method for that type is invoked. To build this part of the interface, we only had to specify the screen relationships between the computers at a site.

Figure 10 also contains a popup window that displays attribute information about a selected computer. Since different computers have different attributes, the information given for a Sun will differ from the information given about a Pyramid. Once again, the database already knows these differences and therefore, much of our work is done for us, especially if there is already a built-in method for displaying attribute information.

4. Implementation

FaceKit is implemented on Sun workstations. It is written in C and runs in a UNIX environment. All the window management is done using Sunviews window package. This package also produces the graphics and handles user input. Since Sunviews manages the windows, they may be moved, hidden, resized, collapsed, etc. just as any other Sun window.

Methods for creating representations are stored in Cactis along with the database objects they represent. No distinction is made between them and regular Cactis objects. Operation descriptions are also in Cactis, but FaceKit creates separate database objects (distinct from regular Cactis objects) for storing them.

5. Future Directions

There are several areas of related research we would like to explore in the future. First, we want to look into giving FaceKit the ability to bootstrap interfaces. An interface designer could then design an interface using interfaces previously designed with FaceKit. For example, Freeform could be used to define representations and operations when building a new interface. Along the same line, we would like see if interfaces not built with FaceKit could be used as tools for building interfaces with FaceKit. This means that some methods for translating random interface queries into a form usable by FaceKit need to be developed. Whether this is feasible to any significant extent, remains to be seen.

There is also the issue of portability. Currently, FaceKit is built on top of Cactis. Is it possible to create a tool like FaceKit that can be easily connected to a wide set of object-oriented databases? Also, can the techniques used by FaceKit be successfully used by other UIMS's? The techniques that depend on extensive knowledge of the application may not translate well to a general purpose UIMS. However, they may prove useful in other special application UIMS's. Identifying these applications and seeing which techniques work well with them could prove to be an interesting research problem.

References

[Bar86] P. S. Barth, "An Object-Oriented Approach to Graphical Interfaces", *ACM Transactions on Graphics 5*, 2 (April 1986), 142-172.

[BoC74] R. Boyce and D. Chamberlin, "SEQUEL: A Structured English Query Language", *Proceedings of the ACM-SIGFIDET*

Workshop on Data Description, Access and Control, May 1974, 219 -261.

[BrH86] D. Bryce and R. Hull, "SNAP: A Graphics-based Schema Manager", *IEEE Conference On Data Engineering*, 1986, 151-164.

[BLS83] W. Buxton, M. R. Lamb, D. Sherman and K. C. Smith, "Towards a Comprehensive User Interface Management System", *Computer Graphics 17*, 3 (July 1983), 35-42.

[Car87] L. Cardelli, "Building User Interfaces by Direct Manipulation", *Digital Systems Research Center Tech. Report*, October 1987.

[Ege88] R. K. Ege, "Defining Constraint-Based User Interfaces", *Data Engineering 11*, 2 (June 1988), 54-63.

[GGK85] K. J. Goldman, S. A. Goldman, P. C. Kanellakis and S. B. Zdonik, "ISIS: Interface for a Semantic Information System", *SIGMOD Conference Proceedings* , May 1985, 328-342.

[Gre84] M. Green, "Report on Dialogue Specification Tools", *Computer Graphics Forum 3* (1984), 305-313.

[Gre85] M. Green, "The University of Alberta User Interface Management System", *Computer Graphics 19*, 3 (July 1985), 205-213.

[Gre86] M. Green, "A Survey of Three Dialogue Models", *ACM Transactions on Graphics 5*, 3 (July 1986), 244-275.

[Gre87] M. Green, "Directions for User Interface Management Systems Research", *Computer Graphics 21*, 2 (April 1987), 113-116.

[HSW75] G. Held, M. Stonebraker and E. Wong, "INGRES: A Relational Data Base Management System", *Proceedings of the AFIPS National Computer Conference 44* (May 1975), 409-416.

[Hen86] D. A. Henderson, "The Trillium User Interface Design Environment", *CHI 86 Proceedings*, April 1986, 221-227.

[HuK86] S. Hudson and R. King, "CACTIS: A Database System for Specifying Functionally-Defined Databases", *Proceedings of the International Workshop on Object-Oriented Databases*, Sept. 1986, 26-37.

[HuK87] S. Hudson and R. King, "Object-Oriented Database Support for Software Environments", *SIGMOD Conference Proceedings*, May 1987.

[HuK88] S. Hudson and R. King, "Semantic Feedback in the Higgens UIMS", *IEEE Transactions on Software Engineering 14*, 8 (August 1988).

[HHN] E. L. Hutchins, J. D. Hollan and D. A. Norman, "Direct Manipulation Interfaces", in *User Centered System Design* , 87-124 .

[KiM84] R. King and S. Melville, "Ski: A Semantic-Knowledgeable Interface", *VLDB Conference Proceedings*, Singapore, August 1984.

[KiN87] R. King and M. Novak, "Freeform: A User-Adaptable Form Management System", *VLDB Conference Proceedings*, Brighton, England, 1987.

[KGM84] H. Kitagawa, M. Gotoh, S. Misaka and M. Azuma, "Forms Document Management System SPECDOQ - Its Architecture and Implementation", *SIGOA Conference Proceedings*, June 1984, 132-142.

[LeL83] A. Lee and F. H. Lochovsky, "Enhancing The Usability of an Office Information System Through Direct Manipulation", *CHI Conference Proceedings*, 1983, 130-134.

[Mye87] B. A. Myers, "Creating Dynamic Interaction Techniques by Demonstration", *CHI + GI* , 1987, 271-278.

[ODR85] D. R. Olsen, E. P. Dempsey and R. Rogge, "Input/Output Linkage in a User Interface System", *Computer Graphics 19*, 3 (July 1985), 191-197.

[Ols86] D. R. Olsen, "MIKE: The Menu Interaction Kontrol Environment", *ACM Transactions on Graphics 5*, 4 (October 1986), 318-344.

[Ols87] D. R. Olsen, "Larger Issues in User Interface Management", *Computer Graphics (ACM SIGGRAPH Workshop on Software Tools for User Interface Management 21*, 2 (April 1987), 134-137.

[SRH85] A. J. Schulert, G. T. Rogers and J. A. Hamilton, "ADM - A Dialog Manager", *CHI 85*, April 1985, 177-183.

[SHB86] J. L. Sibert, W. D. Hurley and T. W. Bleser, "An Object-Oriented User Interface Management System", *Computer Graphics 20*, 4 (August 1986), 259-268.

[SwS88] M. Swain and C. Stepleton, "Design of a Data Definition Language for Cactis", *University of Colorado, Computer Science Dept. Tech. Report*, December 1988.

[SzM88] P. A. Szekely and B. A. Myers, "A User Interface Toolkit Based on Graphical Objects and Constraints", *OOPSLA Proceedings*, 1988, 36-45.

[Ull] J. Ullman, "Principles of Database Systems", Second Edition, Computer Science Press.

[YHS84] S. B. Yao, A. R. Hevner, Z. Shi and S. Luo, "FORMANAGER: An Office Forms Management System", *ACM Transactions on Office Information Systems 2*, 3 (July 1984), 235-262.

[Zlo75] M. M. Zloof, "Query By Example", *Proceedings of the National Computer Conference 44* (1975), 431-438.