Building Reusable Data Representations with FaceKit

Roger King
Michael Novak

Department of Computer Science
University of Colorado
Boulder, Colorado 80309

Abstract

FaceKit is a toolkit for designing interfaces to object-oriented databases. It provides users with a set of tools for building custom interfaces with minimal programming. This is accomplished 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. One of the main features of FaceKit is the ability to rapidly create reusable graphical constructs for representing schema or data objects. These constructs are stored within the database as methods, often dependent on attribute data. Building up a library of these data representations facilitates reusability within a variety of databases.

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

1. Introduction

FaceKit is a window based, interactive graphical system for designing graphics-based interfaces for object-oriented databases. Since we do not support a formal design phase, some people might call FaceKit a tool for *building* interfaces. 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 that they manipulate is built into FaceKit. This knowledge allows for default representations of database objects and for 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 a class of object representations and also leave the defaults in effect where desired.

The main goals of FaceKit are:

- Incorporation of database specific knowledge into a UIMS type tool.
- Encapsulation of interface constructs into the database.
- A unified data model for the interface, and the database.
- Building of a database application and its corresponding interface as an integrated unit.
- Faster and easier design of database interfaces with minimal programming.
- An extensible framework of reusable graphical representations of database constructs.
- Examination of *families* of interfaces designed for a set of related applications.

In this paper, we will explain how FaceKit is used to create a variety of graphical representations of database objects (specifically data objects), and how this affects designing database interfaces in general. Since the incorporation of database specific knowledge into FaceKit allows the interface being built to "share" the database data model and to use the tools provided by the database, it becomes possible to rapidly build a large set of representations for a data object. This also allows for pieces of an interface to be used

This work was supported in part by ONR under contract numbers N00014-86-K-0054 and N0014-88-K-0559 and in part by USWEST under contract number OCG0683B.

for building other interfaces, since everything is accessible through the database. By easily creating a "library" of object representations, an interface designer can create a set of interfaces, each with "tailored" functionality. This helps to avoid having to deal with large all purpose interfaces that are typically unwieldy and perform no single function particularly well. For a more complete description of FaceKit, please see [KiN89].

1.1. Motivation

In the last few years there has been much interest in graphical database interfaces such as ISIS [GGK85], Ski [KiM84], and SNAP [BrH86], which allow schema manipulation in an interactive graphical environment, as well as office forms systems like FORMANAGER [YHS84], Freeform [KiN87], and SPECDOQ [KGM84]. There has also been a whole body of work on general purpose interface creation in the UIMS field. The main emphasis in much of this research has been on dialogue control (the bridge between the interface and the application) [Gre86]. Although there have been systems with dialogue models based on transition networks, grammars, and events, these system share a common perspective. UIMS's such as ADM [SRH85], Grins [ODR85], GROW [Bar86] , Menulay [BLS83], MIKE [Ols86], and Trillium [Hen86] all view an interface as a dialogue between the user and the application. Work has also been done on the gathering of input and presentation of output, including Peridot [Mye87] and Squeak [Car87]. Some other systems have focused on the relationship between the interface and application data model. Filters [Ege88] and Coral [SzM88] each provide a method of specifying relationships between application and interface objects, while GWUIMS [SHB86] and Higgens [HuK88] both allow for sharing of data between the application and the interface. Recently, there has also been some work done on providing a tighter coupling between the application and the interface [WBB90] in order to give the interface more feedback about application data.

There are two main reasons why we feel that general purpose UIMS's do not address our needs. First, although we have seen systems that can communicate with application data in some manner, a general purpose UIMS has no knowledge of database schemas. Second, unlike most interfaces, creating a new database interface often involves creating a new application.

In order to support the desired type of interface, we wish to allow the user to interactively design both an interface and its corresponding application simultaneously. By integrating some database and UIMS technology [Gre87, Ols87], FaceKit treats the interface being designed as an integrated unit with the database, rather than as a dialogue between a distinct user interface and an application. The finished interface will use the database data model and have access to the database schema and to database tools such as query languages and methods. All these tools are also available while using FaceKit to build the interface. This gives the FaceKit 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. Database objects, methods, etc. may also be incorporated directly into the interface, since interface objects have the same structure as database objects and are also stored in the database. This allows for "realistic" default interfaces and faster specification of representations.

1.2. Architecture

FaceKit is built on top of an object-oriented DBMS named Cactis [HuK87, HuK89]. Cactis views an application environment as a collection of *constructed objects*. An object may have *attributes* and *relationships*, both of which are typed. A connector allows a relationship to be applied to a certain object. Connectors may also have *wires* that pass information. For example, figure 1 shows a partial data definition file with object types **country** and **site**. Both have some simple type and complex type (**nametype** and **border**) attributes and at least one connector. The connector of type **country_site** allows for the creation of a relationship between objects of type **country** and type **site**. Attributes may also be functionally derived. For example, the attribute **sites_in** (of **country**), uses the connector **sites** (**sites.w0** is a wire) to iteratively find all instances of type **site** connected to an instance of **country**. The attribute **computers_at** (of **site**) finds all instances of type **computer** connected to an instance of type **site**.

FaceKit uses the data model and the database management tools provided by Cactis for data and schema manipulation. Communication

```
instance type country
        relationships
                plug sites              : country_site;
        attributes
                inst                    : int32;
                name                    : nametype;
                border                  : bordtype;
                new_site_in             : int32 := add_site(inst);
                map_drep                : int32 := draw_map(inst,name,border);
                sites_in                : int32 := iterate tmp : int32
                                                init 0
                                                for each w0 in sites do
                                                        tmp := tmp + sites.w0
                                                end;
        end;

instance type site
        relationships
                socket cntry            : country_site;
                plug to_site            : site_site;
                socket from_site        : site_site;
                plug computers : site_comp;
        attributes
                inst                    : int32;
                site_name               : nametype;
                country_of              : int32 important := cntry.w0;
                loc                     : pairtype;
                computers_drep: int32 := draw_all_comps(inst,site_name);
                map_site_drep           : int32 := draw_site(site_name,loc);
                connect_drep            : int32 := connect_site(inst,loc);
                computers_at            : int32 := iterate tmp : int32
                                                init 0
                                                for each w0 in computers do
                                                        tmp := tmp + computers.w0
                                                end;
        end;
```

Figure 1

with these Cactis tools and coordination of interface tasks is done by the representational and operational components. The primary responsibility of the representational component is managing the visual representations of database objects and I/O while 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.

Since this paper concentrates on representations of data objects, we are only going to discuss the representational component. For the purpose of this discussion, database object, or object, may refer to a constructed object, relationship, or attribute. The representational component builds, maintains and invokes the methods used to produce the visual representations of objects. Not only does this make the 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 without making representations created earlier

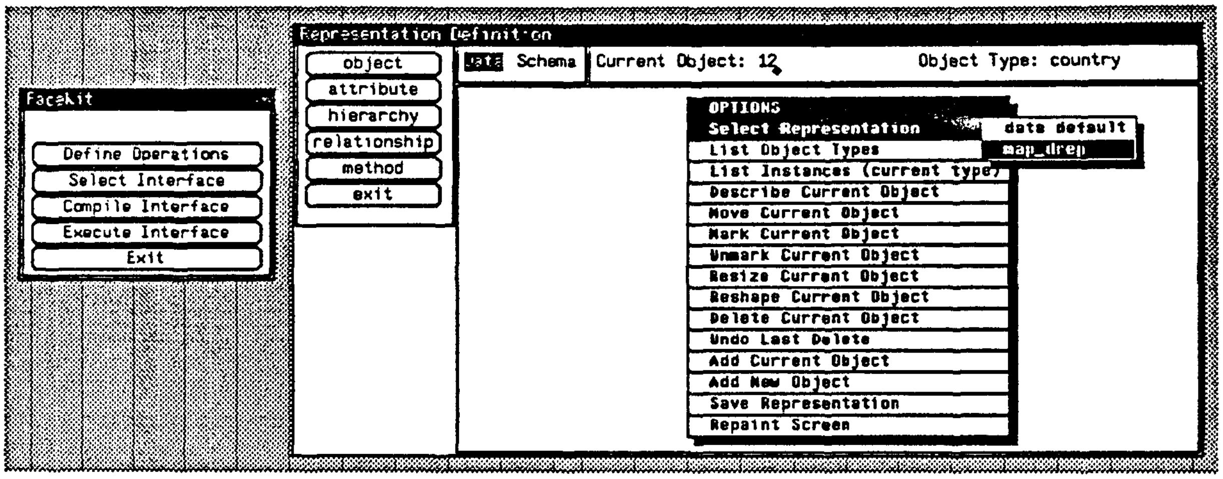

Figure 2

obsolete.

2. Designing FaceKit Data Representations

The FaceKit approach to designing interfaces is somewhat different from the common UIMS approach. Traditionally, UIMSs have allowed the user 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. Our approach allows the user to define what we view as two somewhat different aspects of the interface: appearance and functionality. It is the appearance part in which we are interested here, specifically the appearance of data objects.

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 could even 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 instance of type **country**, the interface uses a previously defined representation of **country**.

2.1. Using FaceKit Data Representations

To illustrate how representations are defined and used, we examine an example that describes CSNET sites. Figure 2 shows a representation definition window. Currently, the user is defining data representations, however, the same interface is used to define schema representations. The object instance being viewed is of type **country** (instances can be listed by selecting **List Instances** from the menu). The two representations for an instance of type **country**, available at this time, are the default and **map_drep**. The default invokes a method that provides a listing of each attribute, along with its type and value. This method is inherited from the "universal" class that all classes are a subset of. **Map_drep** is a user-defined representation that is stored as an attribute of the type **country**. The user in figure 2 selects **map_drep**. The result of this selection is shown in figure 3.

This representation is a drawing of a particular country and some of the CSNET sites in that country. It is created by associating a method named **draw_map** with the attribute called **map_drep**. The method is invoked by referencing the attribute **map_drep**. The data that this

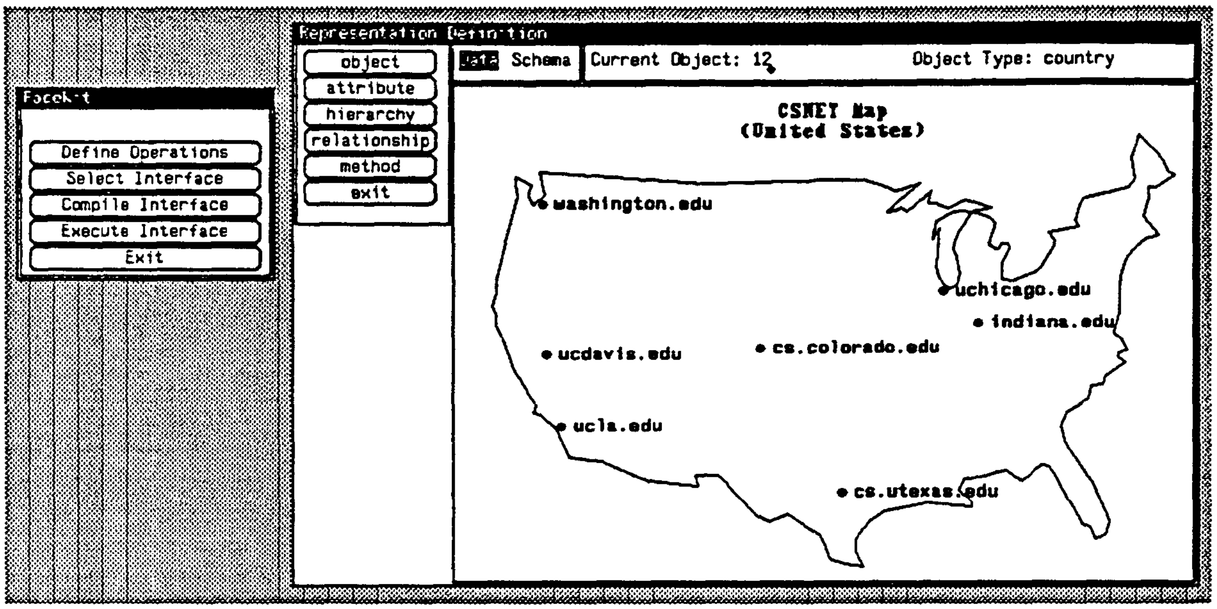

Figure 3

method needs in order to draw this representation is also stored in other attributes of the type **country**. This encapsulation allows FaceKit to easily keep track of the various representations available for a data type. Since FaceKit keeps track of the representations available for various object types, a library of data representations can quickly be built. Not only does this make it easy to build an interface, it also allows for reuse of the data representations when building multiple interfaces.

The availability of default representations, combined with the reusability of user created representations, makes FaceKit more interactive than many systems. Although code has to be written to initially create data representations, they can then be "plugged into" an interface interactively. Also, representations may be built up from other representations that already exist, rather than having to be built from scratch.

2.2. Storing Data Representations

FaceKit data representations are stored as reusable methods dependent on some set of object attributes rather than as simple bitmaps. Therefore, although a representation of an object could be as simple as an attribute that is merely a bitmap, this is not the way that most data representations would be done. For example, the way that the map shown in figure 3 is stored and drawn can be seen by examining the section of a data definition file shown in figure 1. The method for drawing the map is connected to the attribute **map_drep** and dependent on the attributes **inst**, **name**, and **border**. **Border** is the attribute that contains the actual locations for drawing a border for the given country. These attributes never need to be explicitly passed to the method because they are defined in the schema and thus encapsulated in an object type. Therefore, each method already knows what its arguments are.

The sites in the map are drawn by following the **sites_in** attribute to the appropriate set of site objects. Each site is drawn by the representation method attached to the **site** attribute **map_site_drep**. This "layered" representation approach allows other representation methods to also use **map_site_drep** if desired. Also, a site could be included within many different maps and no new representations need be developed for it. Note that there could be many different representations available for a site. It would then be up to the person developing a map drawing method to choose which one they wanted to use.

Another important aspect of the storage method used is the transparent integration of new and old data representations. All the data representations created previously are also available to a user and look no different than newly created ones. These previously created representations may be used just like new ones for building new interfaces. This is a key factor in providing representation reusability.

3. Conclusions

The main objective of FaceKit is to explore ways of rapidly and easily providing new interfaces to object-oriented databases. One of the necessities for accomplishing this, was to provide tools for creating and maintaining data representations. This capability has proven to be an interesting research area of its own. We have found that not only is the ability to create a library of representations important to FaceKit, it is also a good way to view database representations in general.

By taking this approach, a designer will have some building blocks for creating database interfaces. Also, since the representations are all stored within the database and transparent to the user, they are easy to maintain. Since the structure of the representation data is compatible with other database data, maintaining consistency as the database changes is kept as simple as possible. This compatibility also makes it possible to use a query language to manipulate data representations in the same manner as any other database objects.

References

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

[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.

[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.

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

[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).

[HuK89] S. Hudson and R. King, "Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System", *ACM Transactions on Database Systems 14*, 3 (Sept. 1989), 291-321.

[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.

[KiN89] R. King and M. Novak, "FaceKit: A Database Interface Design Toolkit", *VLDB Conference Proceedings*, Amsterdam, The Netherlands, August, 1989, 115-123.

[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.

[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.

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

[WBB90] C. Wiecha, W. Bennett, S. Boies, J. Gould and S. Greene, "ITS: A Tool for Rapidly Developing Interactive Applications", *ACM Transactions on Information Systems 8*, 3 (July 1990), 204-236.

[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.