A RETROSPECTIVE ON DATABASE APPLICATION DEVELOPMENT FRAMEWORKS

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M92/13

23 January 1992

A RETROSPECTIVE ON DATABASE APPLICATION DEVELOPMENT FRAMEWORKS

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M92/13

23 January 1992

TITLE PAGE

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A RETROSPECTIVE ON DATABASE APPLICATION DEVELOPMENT FRAMEWORKS

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M92/13

23 January 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Retrospective on Database Application Development Frameworks

Lawrence A. Rowe

Computer Science Division - EECS
University of California at Berkeley
Berkeley, CA 94720
(Rowe@CS.Berkeley.EDU)

Abstract

Four application framework models developed by the author for database application development systems are described. The key feature of these systems is to provide a model for the definition of high level objects that represent interface abstraction that can be used to build an application. Structuring application code around interface objects reduces the conceptual distance between the executing program and its specification. At the same time, good programming practices must be supported (e.g., code modularity, reusable components, and information hiding).

1. Introduction

This paper presents a retrospective on what I have learned during the past 15 years on the design and implementation of development tools for database applications. During this time I directed or consulted on five systems for database application development:

(1) The **Rigel** programming language developed between 1976 and 1979 at U.C. Berkeley [Rowe79].
(2) The Forms Application Development System (FADS) developed between 1979 and 1982 at U.C. Berkeley [Rowe82, Shoens82].
(3) Application-By-Forms (ABF) developed between 1983 and 1986 by Ingres Corporation [Ingres90, Rowe85].
(4) The **PICASSO** system developed between 1987 and 1991 at U.C. Berkeley [Rowe91].
(5) Windows/4GL (W4GL) developed between 1988 and 1990 by Ingres Corporation [Ingres91].

Rigel was a Pascal-like database programming language. It provided a traditional character stream-oriented input/output system that made it impossible to develop forms-based applications. **Rigel** itself did not have a major impact but many ideas developed in that system were used again in later systems.

FADS and ABF were designed to develop forms-based applications that would run on an alphanumeric terminal, and **PICASSO** and W4GL were designed to developed graphical applications that would run on a bit-mapped terminal or workstation. These four systems have a higher level application framework that simplifies application development.

This paper describes the evolution of the application framework model from FADS to W4GL.

2. FADS Application Framework

An application framework model represents an application by a collection of objects that encapsulate the data abstractions, user interface, and application behavior. The major difference between the models was the type of objects supplied and their behavior. FADS supported the following object types:

(1) *Relations* stored in the database.
(2) *Datatypes* that represented values such as, integers, strings, and dates.
(3) *Procedures* that contain application code.
(4) *Forms* that contained fields through which data is displayed to and edited by the user.
(5) *Frames* that contain a form and a menu of operations the user can execute. An operation can change the user interface (e.g., display the next employee) or invoke an application procedure (e.g., create a new product number in a bill-of-materials).

Frames are similar to procedures in that they have local variables and they can be called. Frame variables are bound to fields in the form so the value displayed to the user is always the current value of the variable. Frames also have parameters so that values can be passed to them when they are called. Formal parameters are local variables so it is very easy to display data to the user. The application calls a frame and passes it the data. The system displays the form and menu to the user. The data passed to the frame is displayed in the field to which the formal parameter is bound.

†This research was supported by the National Science Foundation under grant MIP-90-14940.

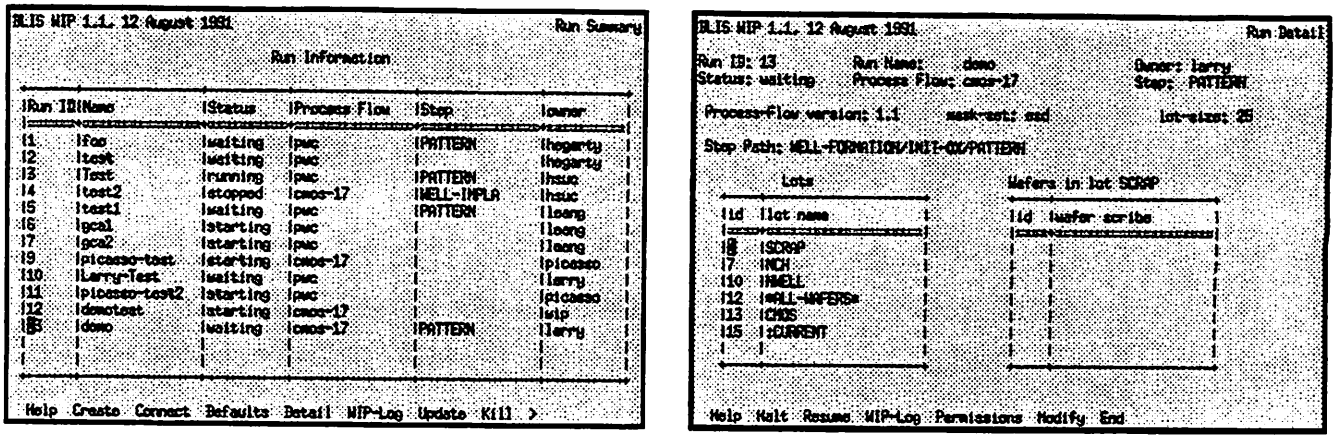

Figure 1: Two frames from a shop floor control system.

The form and operation menu are displayed to the user in a standard way, and the system provides built-in commands to enter and edit data. Consequently, the application developer does not have to write code to display the form and interact with the user. The developer designs the form with a direct manipulation editor and specifies the operations. Figure 1 shows two frames from a shop floor control system used in semiconductor manufacturing. The frame on the left lists active runs. When the user selects a run and executes the *Detail* operation, the frame on the right replaces the frame on the left on the user's screen.

FADS treated forms as a separate object so they could be reused in different frames or forms. For example, a name and address block with edit checks and operations can be defined as a form and reused in other forms to standardize the way they are displayed to and edited by the user. Datatype objects included display attributes (e.g., edit checks, input masks, and default values) so that forms defined by specifying the datatype displayed through a field would have the default display attributes.

FADS had a direct manipulation interface editor that allowed users to define objects by filling in forms. In addition, the forms system had table fields which are an essential interface widget for applications with structured data (e.g., tables, sets of objects, etc.). A table field displays a record in each row.

The major problem with FADS was that it was too slow, and I had trouble getting the funding agencies to continue work on a system that seemed indistinguishable from the screen painters being developed in industry. The major performance problem was caused by the way applications were stored in the database. The complete application specification including forms, types, and 4GL code was stored in the database. We fully normalized the database design so that each statement in an operation and each field in a form was stored in a separate record. As you might expect, it took a long time to fetch a frame definition from the database. Nevertheless, the FADS prototype demonstrated that this approach to building applications made sense.

3. ABF Application Framework

I co-founded Ingres Corporation in 1980 with Michael Stonebraker and Eugene Wong and by 1982 it was clear the company needed a 4th Generation Language system. I was able to convince them that a commercial product with a FADS-like model would be a good product. The resulting product was ABF.

ABF introduced several new concepts to the model: frame types, popup forms, and global variables. FADS provided only one type of frame. The developer had to specify the form and the operation code. ABF supplied report and Query-By-Forms (QBF) frames so an application developer did not have to specify as much detail. For example, report frames are defined by specifying the report. The system automatically supplies a form with fields to enter the report parameters and operations to display the report on the terminal or send it to a printer. QBF frames supply operations to query and update data through a default form generated from the database schema or a custom-designed form specified by the developer. In essence, frames are created by application generators integrated into the development environment.

Because FADS and ABF ran on alphanumeric terminals, they displayed only one frame at a time. When an application called another frame, the current frame was replaced by the called frame. When that frame returned, the previous frame was redisplayed. ABF added the concept of *popup forms* which are displayed in a small window on top of the current frame. Popup forms are typically used to dis-

play acceptable values for a field. They were better than calling another frame because the rest of the current frame was still visible. In fact, popup form is really a misnomer. Since the popups received the keyboard focus and contained operations, they were really frames. However, it was more convenient to specify the popup as part of the form in which it would be used rather than define a separate frame and call it.

ABF also added global variables and local variables not bound to form fields. In FADS, all variables were local and they were all bound to form fields. This change was the first of many that added a programming language to ABF. My initial concept for the FADS 4GL was that operations would be specified in an extended query language, not a programming language. The goal was to create an end-user programming environment that did not require significant programming experience. A conventional 3GL procedure could be called from the 4GL so the user could write a complex procedure if required, but he or she was not forced to do so. Unfortunately, this approach did not work because the development environment made it difficult to edit and recompile 3GL procedures and users wanted the power of a full-function programming language. Over time the ABF 4GL has evolved into a reasonably complete programming language. In fact, many of the procedural constructs in the ABF 4GL were modelled on the language constructs developed in Rigel.

The ABF model was not perfect because nested forms and type objects were omitted, and developers could not define new frame types. They were omitted to reduce the project complexity and development time. Nested forms and type objects were missed, but they are not as important as user-defined frame types.

ABF supported three frame types: user, report, and QBF. The system needed menu frames, that is frames that show a list of operations and documentation for each one. Many users developed standard menu frames, but they were difficult to specify in an application.[1] If the system supported user-defined frame types, developers could have built menu frames into the system themselves.

Notwithstanding these limitations, ABF was a very successful product. For several years ABF and the other forms-based interface products from Ingres Corporation were considered the best database tools in the industry.

A static analysis of 30 ABF applications was performed to see how people used ABF [Gardner88]. The applications were taken from 3 companies. The largest application contained over 15,000 lines of code. The study showed that the average application was composed of 30 frames and that 25% of the forms were used in more than one frame. The study also found that on the average user frames contained 5 operations and each operation contained 10 lines of code. Another interesting result was that applications were either composed predominately of user frames or they were composed of QBF and report frames. The applications composed primarily of user frames were production applications that typically included many application-specific operations (e.g., release purchase order). The other applications were what I call *ad hoc applications* developed by end-users to solve an immediate need. These applications often bring real value to an organization and over time they become production applications in the sense that the organization cannot run without them.

[1] Several third party software companies and consultants developed flexible menu systems that they sold as products or used to improve their own productivity.

4. PICASSO Application Framework

In late 1985, I decided to build a system to develop graphical user interface (GUI) applications. The system, called **PICASSO** was begun in early 1986 and released outside Berkeley in late 1989. The ABF model had to be modified to allow multiple windows to be displayed at the same time and to include other common GUI interface abstractions. Two new interface objects were created: *dialog boxes* and *panels*. A dialog box is a modal interface that is used to confirm an operation, collect further arguments for an operation, or report an error. A panel is a non-modal interface that is typically used to present more detailed information about an entity (e.g., selecting an employee in a list might show more details about the employee in a panel) or the same information in a different representation.

The behavior of dialog boxes and panels is different than frames. Whereas a frame is like a procedure, a dialog box is like a function. That is, the dialog box is called, the user is forced to respond to it, and a result is returned to the caller. It is positioned at the center of the currently active frame or panel. A panel is like a co-routine. It is displayed to the user when the panel is first called, but execution of the panel is suspended when the user moves the mouse cursor outside the panel window. The panel execution is resumed when the mouse is moved back into the window.

Each framework object, called a *PICASSO object (PO)*, has a different visual appearance. A frame has a menubar with pulldown menus across the top of its window. A panel can either have a menubar across the top or buttons listed down the right side of the window. Frames and panels have title bars provided by the window manager. Dialog boxes have buttons down the right side and no title bar since they are non-modal interfaces.

Figure 2 shows a screen dump of a semiconductor CIM database browser [Smith 90]. It has a frame and three panels. The frame in the upper left corner of the screen shows the facility floorplan, the panel in the upper right corner shows equipment in the facility, the panel in the lower right corner shows utility lines running through the facility, and the panel in the lower left corner shows a picture of a particular piece of equipment.

PICASSO introduced lexical structure between PO's to solve two problems: data sharing and window hiding. PO's can share data either implicitly by accessing variables in a common parent or explicitly by passing param-

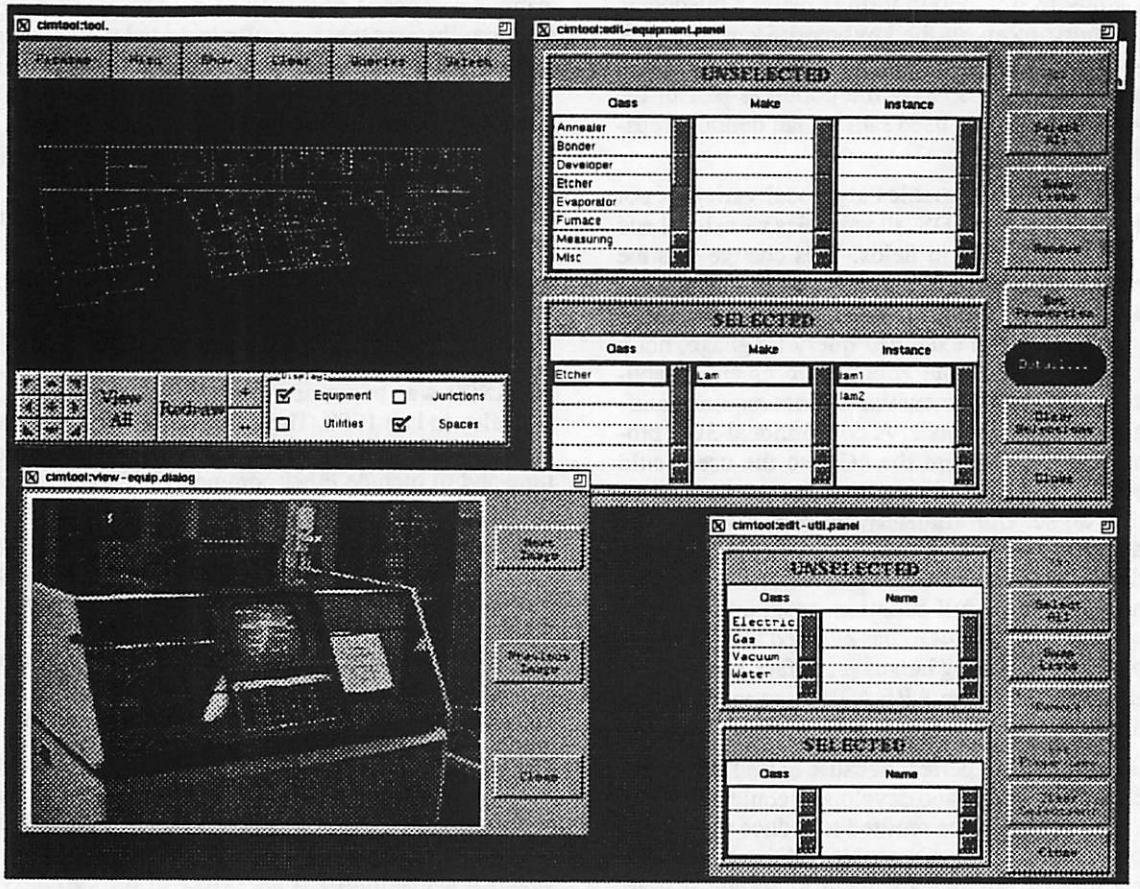

Figure 2: CIM database browser.

eters. Both approaches have proven convenient. The second problem lexical structure solved is when to hide a PO. When a frame is called or exited, the current frame and all visible children of that frame are hidden. Suppose that the lexical parent of the equipment picture panel in figure 2 is the equipment panel in the upper right corner, and the parent of the equipment panel is the frame. Consequently, when an operation returns or exits the equipment panel, the equipment picture panel is removed too. Declarative specification of these relations based on lexical structure is easier for the programmer to understand than procedural specifications.

PICASSO was written in Common Lisp using the Common Lisp Object System (CLOS) and the X Windowing System. The 4GL is essentially Lisp with extensions to call PO's, access variables, and execute database queries. I would have preferred to develop a simpler end-user 4GL, but resources were limited.

The use of Lisp was both positive and negative. On the positive side, it is a great prototyping environment particularly for experimenting with language constructs. We were able to implement a complete constraint system that allows us to modify variables or interface behaviors as the result of functions of other variables in less than 3K lines of code. On the other hand, we have struggled to build space and time efficient production applications. Another problem we have had is attracting and training people to use the system. It takes 4-6 months to train a competent Lisp programmer and another 6 months for them to learn the body of the **PICASSO** system. We have built a prototype interface builder, but it still needs considerable work. And we are running into the same funding problem we had with FADS because people think that the Next Interface Builder and Hypercard have solved all problems.

5. W4GL Application Framework

In late 1988, I convinced the folks at Ingres to build a next generation development environment for GUI applica-

tions to replace ABF. W4GL simplified the PICASSO model by reducing the number of distinct callable interface objects. It has only one object, called a frame or window, rather than the three objects in PICASSO (e.g., frame, dialog box, and panel). However, there are two frame types: menu and dialog. A menu frame corresponds to a PICASSO frame and a dialog frame corresponds to a dialog box. Panel behavior is specified either in the frame definition or in the statement that calls the frame.

At the time, I thought this approach was a poor design choice because the visual appearance of the application interface did not clearly delineate the behavior of the different windows. A programmer could create two windows that looked exactly alike, but behaved differently. The user could not determine which windows were frames and which were panels. The early design did not distinguish dialog boxes, although they were eventually separated out. In retrospect, this approach probably is not a problem since we were forced to add menubars to panels in PICASSO which made them indistinguishable from frames. However, we still use buttons down the right side for most panels.

W4GL extended the ABF 4GL to include an object system. The entire system was written in this object system. Numerous people have noted that an object system language is the best interface toolkit and application implementation language. The object system in W4GL uses a single inheritance hierarchy and methods that discriminate on one argument (i.e., unimethods). The CLOS system we used for PICASSO uses multiple inheritance, uni- and multi-methods, and method combinations. Our experience was that multiple inheritance was useful, multi-methods were unnecessary, and method combinations were useful, but difficult to use and slow [Konstan 91]. One of the W4GL implementers remarked that multiple inheritance would have simplified the implementation of the forms system on which the system runs. Consequently, I believe multiple inheritance is a good idea.

The biggest improvement in W4GL, aside from the support for GUI interfaces and the direct manipulation interface builder, was the addition of a version control system on application objects. From the very beginning in FADS, it was clear that a version control system was needed so that multiple programmers could work simultaneously on objects in the same application. W4GL is the first system to provide one, and I believe it is still the only interface builder in the market today with a built-in version control system. Real world applications include many objects and multiple versions, and you need help from your development environment to manage this complexity.

The primary problems with W4GL are that it does not allow users to add new widgets to the interface library (e.g., 3D graphics, video, and audio widgets), application generators were omitted, there is no general constraint system, the 4GL object system does not allow users to define subclasses, and it does not support persistent objects. Presumably, these problems will be fixed in future releases.

6. Thoughts on the Past and Future

An object-oriented application framework with a direct manipulation interface builder and application editor is the only way to develop database applications. A high level framework simplifies the definition of applications because less code must be written and custom direct manipulation editors can be developed for each object type.

The systems described here show the evolution in my thinking about the features required in a framework. Each system had many positive characteristics. However, there are still many problems to be solved. First, we need more work on application generators. The basic idea is to build applications at a higher level by configuring subsets of the objects that make up the application with a customized direct manipulation editor. We first experimented with this idea in ABF. The only other system I have seen with integrated application generators is the PACE system from Wang. One goal for the PICASSO was to develop an open system so we could experiment with specific application generators and with "application generator" generators. For example, a company might want to build a custom-designed report frame generator that used the same report formats and frame operations.

Application generators offer great hope for significantly improving programmer productivity because they reduce the specification required to build an application. They must be integrated with the development environment so the custom extensions needed by real world problems can be made.

Second, we still do not have the right abstractions for the interface objects and the 4GL. PICASSO and W4GL made dramatic progress in the GUI application framework, but they still have problems. One good feature of the PICASSO implementation was that the objects were implemented in the 4GL so new interface objects can be added to the system by users. For example, another object type is a windoid which is a non-modal popup that waits for a mouse event but does not grab the keyboard focus.

The future is end-user programming and none of the 4GL's I have seen are easy enough to use. We need to make them easier to learn and improve programmer productivity.

Finally, we need more work on interface builders. No human factors experiments or even pilot studies have been done to compare different interface builders. Interface builders are *essential* components of any modern programming environment because the majority of programs being written will have graphical user interfaces. We need to understand what features contribute to productivity, what features are error prone, and what productivity gains are provided by different programming environment tools (e.g., structured editors, application generators, etc.).

A lot has been learned in the past 15 years, but there are still many exciting challenges ahead. It is time to begin development on the next system!

7. Acknowledgments

Many people have worked on the systems described above. I do not have space to acknowledge all of them, but I do want to acknowledge the principal contributors. Kurt Shoens implemented **Rigel** and designed and implemented FADS. Joe Cortopassi also worked on **Rigel**, was the chief architect and implementer of ABF, and implemented the W4GL runtime system and translator. John Newton implemented QBF. Peter Schmitz worked on ABF and was the project manager for W4GL. Dave Martin and Scott Luebking worked on an early version of **PICASSO**. The design and implementation of the current version of **PICASSO** was done by Joe Konstan and Brian Smith. Steve Langley implemented the forms system in W4GL and Grant Crossen implemented the interface builder.

8. References

[Gardner88] L.Gardner, "Static Analysis of a Fourth Generation Language," MS Report, Computer Science Division - EECS, U.C. Berkeley, June 1988.

[Konstan91] J.Konstan and L.Rowe, "Developing a GUIDE Using Object-Oriented Programming," *Proceedings OOPSLA 1991*, Phoenix, AZ, October 1991.

[Rowe79] L.Rowe and K.Shoens, "Data Abstraction, Views and Updates in **Rigel**," *Proceedings 1979 SIGMOD Conference*, Boston, MA, June 1979.

[Rowe82] L.Rowe and K.Shoens, "A Form Application Development System," *Proceedings 1982 ACM SIGMOD Conference*, Orlando, FL, June 1982.

[Rowe85] L.Rowe, "Fill-in-the-Form Programming," *Proceedings 11th International Conference on Very Large Databases*, Stockholm, Sweden, August, 1985.

[Rowe91] L.Rowe, et.al. "The **PICASSO** Application Framework," *Proceedings 1991 ACM Symposium on User Interface Software and Technology*, Hilton Head, SC, November, 1991.

[Shoens82] K.Shoens, "A Form Application Development System," Ph.D. Dissertation, Computer Science Division - EECS, U.C. Berkeley, November 1982.

[Smith90] B.Smith and L.Rowe, "An Application Specific Ad Hoc Query Interface," ERL Report M90/106, U.C. Berkeley, November 1990.

[Ingres90] INGRES ABF (Application By Forms) User's Guide, Ingres Corporation, Alameda, CA, June 1990.

[Ingres91] Application Editor User's Guide for INGRES/Windows 4GL, Ingres Corporation, Alameda, CA, June 1991.