OBJECT FADS PROJECT STATUS REPORT

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M87/20

16 April 1987

COVER PAGE

OBJECT FADS PROJECT STATUS REPORT

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M87/20

16 April 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

OBJECT FADS PROJECT STATUS REPORT

by

Lawrence A. Rowe

Memorandum No. UCB/ERL M87/20

16 April 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Object FADS Project Status Report[†]

Lawrence A. Rowe
Computer Science Division - EECS
University of California
Berkeley, CA 94720

1. Introduction

Object FADS is a "what you see is what you get" (*WYSIWYG*) programming environment for menu-based, graphical applications that access large shared databases. It will be the programming interface to POSTGRES, a next generation relational database management system currently being developed at the University of California, Berkeley [StR86].

This report summarizes the status of the Object FADS system as of April 1987. The system design has been completed and the implementation has been started. As part of the implementation effort, we have built a Common Lisp interface to the "X Window System" that is being distributed to others [Mar87].

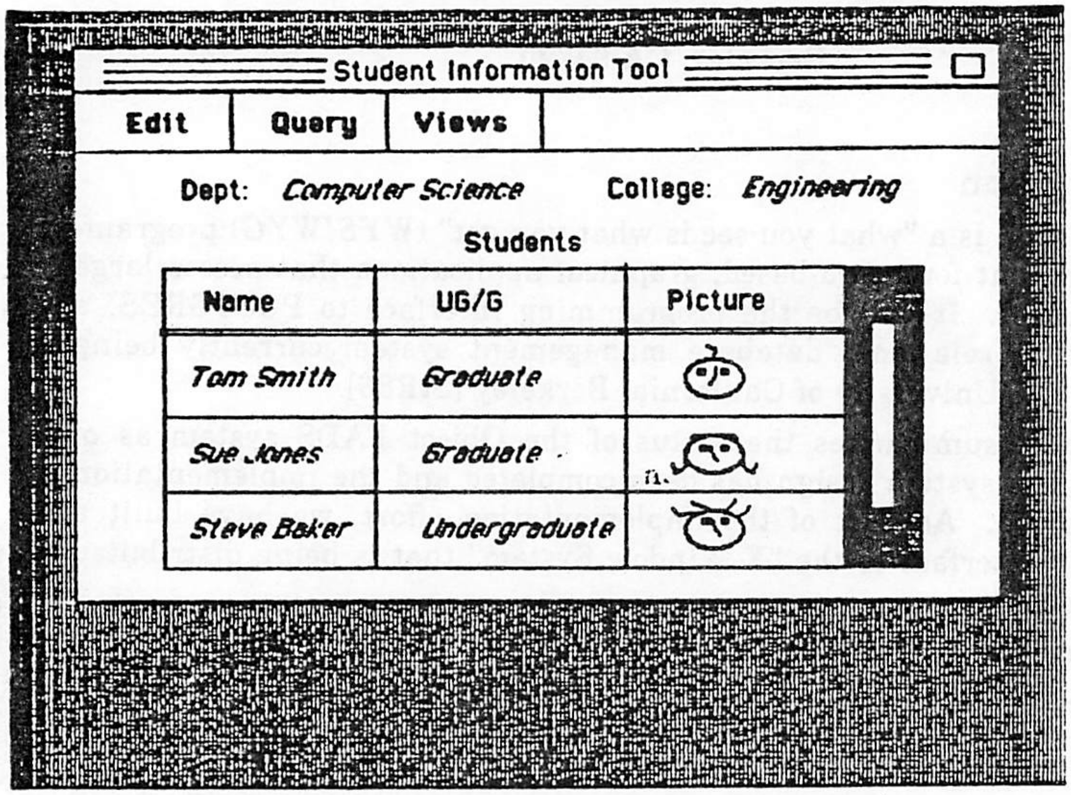
The remainder of this report describes the Object FADS system, the current status of the implementation, and our future plans.

2. Object FADS

Object FADS is an object-oriented programming environment that will allow rapid development of sophisticated, interactive database applications (e.g., a software bug reporting and maintenance system, an IC fabrication process control system, or an office automation system). The applications developed with FADS and the programming environment run on a high performance Unix-based workstation with a bit-mapped display and a mouse. Our goal is to build a portable system that can be run on a variety of Unix-based workstations (e.g., Sun's and DEC microvax's).

The user interacts with an Object FADS application, called a *tool*, through windows displayed on the workstation screen. Figure 1 shows a simple student information tool. A tool is composed of a collection of *frames*. A frame contains a form and a menu of operations. Users interact with the tool by moving between the frames viewing and updating data displayed

[†] This research is supported by the National Science Foundation under Grant NSF-DCR-8507256.



Dept: *Computer Science* College: *Engineering*

Students

Name	UG/G	Picture
Tom Smith	*Graduate*	
Sue Jones	*Graduate*	
Steve Baker	*Undergraduate*	

Figure 1: An example tool.

through the form and executing operations.[1] Operation menus are displayed in a "menu bar" below the tool title (e.g., *Edit*, *Query*, and *Views* are the menus in the student information tool). The operations are coded in a very high-level language with constructs to display data to the user through the form, to allow the user to update the displayed data, to query and update the database, to perform computations, and to call other frames. A tool is analogous to a program in a conventional programming environment and a

[1] Other user-interface development systems use the term *views* or *DisplayObjects* to describe the specification of how to display data to the user. We chose the term *form* because end-users are familiar with forms and their behavior and because the term *view* has a different meaning in database terminology.

frame is analogous to a procedure in a conventional programming language. Several tools can be running at the same time and they may be displaying the same data in different forms (e.g., as text or as an icon).

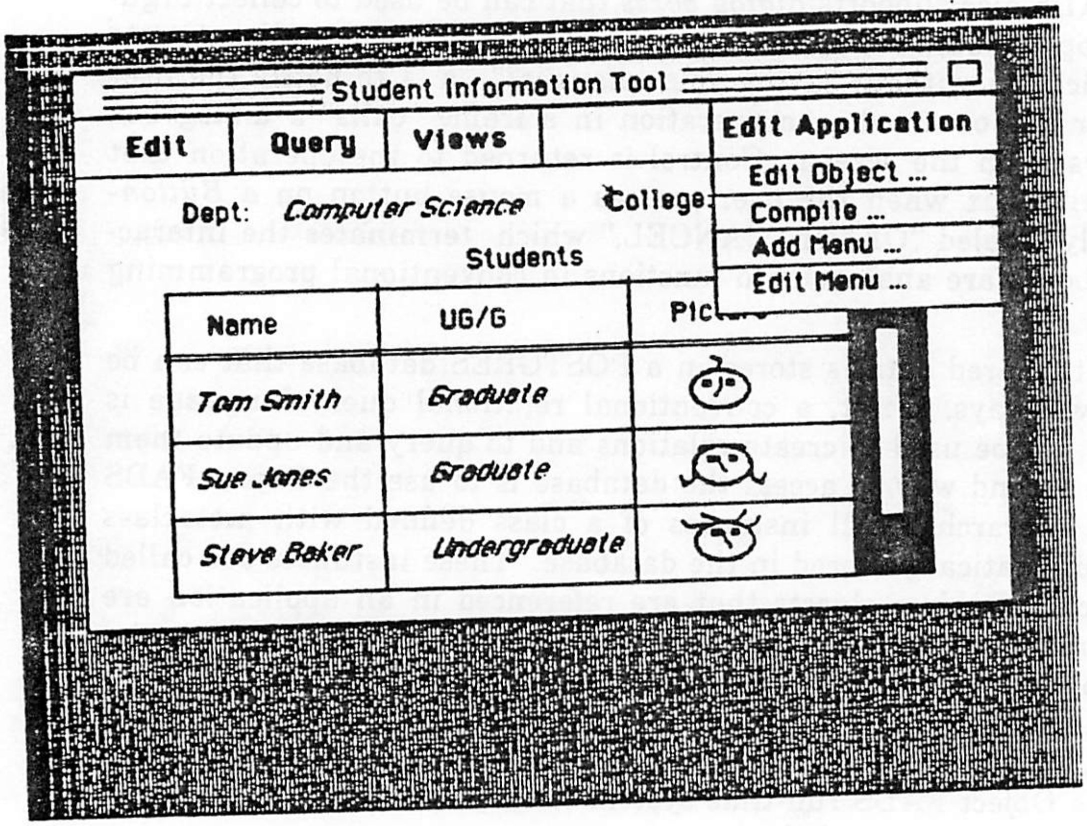
Arbitrarily complex data displays can be created with the Object FADS forms system. It will be possible to present data to the user in different representations (e.g., text, graphics, video, or audio). Primitive form-types (e.g., *CharForm*, *GraphicsForm*, etc.) are provided by the system. In addition, structured form-types are provided that can be used to create more complex forms (e.g., "table" forms, "master/detail" forms, "cross-tabulation" forms, "spreadsheet" forms, etc.). The form in the student information tool in figure 1 contains a scrollable table with columns that display text (e.g., "Name") and graphics data (e.g., "Picture"). The forms system is extensible so that users can define their own primitive and structured form-types.

Object FADS also supports *dialog boxes* that can be used to collect arguments for an operation, to confirm an operation (e.g., "Do you really want to exit this application without saving your changes?"), a d to notify the user when an error has occurred. An operation in a frame "calls" a dialog box which displays it on the screen. Control is returned to the operation that called the dialog box when the user presses a mouse button on a *ButtonForm*, typically labeled "OK" or "CANCEL," which terminates the interaction. Dialog boxes are analogous to functions in conventional programming languages.

Persistent, shared data is stored in a POSTGRES database that can be accessed in two ways. First, a conventional relational query language is provided that can be used to create relations and to query and update them [RoS87]. The second way to access the database is to use the Object FADS shared object hierarchy. All instances of a class defined with metaclass *dbclass* are automatically stored in the database. These instances are called "*dbclass* objects." *Dbclass* objects that are referenced in an application are implicitly retrieved from the database into an "object cache" in the front-end application program. Updates to *dbclass* objects are automatically propagated to the database and other applications that have the object in their "object cache." Concurrency control and crash recovery protocols are implemented by the Object FADS run-time system to control shared access and to protect the data from system failures.

The shared object hierarchy will control the mappping between the Common Lisp and POSTGRES type systems. POSTGRES user-defined data types will be mapped to Common Lisp types and a selection of useful Common Lisp types (e.g., lists) will be mapped to POSTGRES data types. The proposed implementation of the shared object hierarchy is described elsewhere [Row86].

The Object FADS development environment is composed of a collection of editors for the objects that make up an application (e.g., tools, frames, forms, operations, dialog boxes, and objects). An application is defined by describing the objects that compose it. WYSIWYG editors are provided for each object type (e.g., a WYSIWYG forms editor, a WYSIWYG frame editor, a WYSIWYG object editor, etc.). Figure 2 shows the student information tool discussed above with the additional menu item for application development. The operations in the *Edit Application* menu allow the developer to inoke an object-specific editor, to compile the application, and to add, delete, or modify a menu operation. Notice that the developer can easily switch between defining the application (i.e., executing operations in the *Edit Application* menu) and executing it (i.e., executing operations in the other menus).



Dept: *Computer Science*

College:

Students

Name	UG/G	Pic
Tom Smith	*Graduate*	
Sue Jones	*Graduate*	
Steve Baker	*Undergraduate*	

Figure 2: Application development interface.

Frames are specified by defining a form and coding the operations in a high-level programming language [RoS82,RTI84] or by using a frame-generator [Row85,RTI84]. Currently, the operations are coded in Common Lisp which is the language used to implement Object FADS. Functions for high-level operations (e.g., CALL-FRAME and CALL-DIALOGBOX) have been added to the system to simplify operation coding. We plan to explore graphical notations for the operation language in the future.

The second way to define a frame is to use a frame-generator. An example of a frame-generator is a "query/update through forms" frame-generator that takes a form definition and a specification of a mapping between the database and the form and produces a frame that can be used to browse the database. The frame-generator mechanism is also extensible. In other words, developers can create new frame-generators and editors that can be used by end-users to create applications. These frame-generators and the form-types mentioned above are defined by using a frame- or form-definition tool. In other words, the definition environment uses the same interface as the applications that are being built.

The Object FADS system is built on an object-oriented programming system. The objects are stored in a database so that users can share both the data being accessed and the programs that access it. Consequently, new application editors and browsers can be built using the system itself and, if necessary, an application developer can access the low level primitives used to implement the system.

Object FADS has several novel features when compared with other database and expert system programming environments. First, the shared object hierarchy implemented on a relational database management system allows users to code applications in an object-oriented programming language and access data created by applications coded in other conventional programming languages (e.g., ADA, C, COBOL, FORTRAN, or PASCAL)

Second, the extensible forms system and frame-generators coupled with the WYSIWYG editors makes Object FADS a powerful, yet easy-to-use system to build user interfaces. Our experience is that most people underestimate the number of lines of code required to build a sophisticated, user-friendly interface in a workstation environment. Often, more than 50% of the lines of code in an application is really user interface code. Moreover, past experience has shown that with a visual programming environment and high-level application objects (e.g., default forms and frame-generators) novice programmers can build many useful applications and expert programmers can produce sophisticated applications more quickly.

Third, although not part of our current work, our future plans include investigating:

1) graphical notations for specifying operations,

2) the problems of porting the system to a tightly-coupled multiprocessor workstation,

3) application program support for rules defined in the database [SHH87] similar to the support currently provided for objects (i.e., building an integrated model for rules that are executed in the application front-end or database back-end process), and

4) debugging support for rules executed on a tightly-coupled parallel processor.

Finally, Object FADS is the primary programming environment for all of the features in POSTGRES (e.g., abstract data types, historical data, versions, procedure data types, and rules).

3. Implementation Status

After evaluating many different programming languages and environments, we decided to implement Object FADS in Common Lisp. We have built a Common Lisp interface to the "X Window System" [Get86] that we are distributing to others. This system provides access to all functions and data structures in the X Library for C [Gee86].

We are using the "Common Lisp Object System" (CLOS)[2] for the object hierarchy model in Object FADS. Using an implementation of CLOS developed at Xerox PARC, we have defined low-level window system objects (e.g., display, window, font, event, etc.) and primitive versions of some high-level Object FADS objects (e.g., tools, frames, and forms).

We have also completed a low level interface to POSTGRES that allows arbitrary POSTQUEL commands to be executed from Lisp. This library also provides support for *portals*, a new application program interface developed at Berkeley [StR84,StR86]. A simple forms-based relation browser has been built using this library and the form objects.

Lastly, we have built a simple object hierarchy browser that allows us to examine and update CLOS objects. The object hierarchy is displayed graphically and menu operations are provided to examine a class definition (e.g., instance variables, class variables, methods, and superclasses). We are currently modifying it to be an Object FADS tool and extending it to allow the programmer to edit class definitions.

[2] CLOS was formerly known as Common LOOPS [Boe86].

4. Future Plans

This section presents our implementation plans for the next year. Our major task is to complete the basic system and to develop some applications using it. There are three subtasks that must be finished to complete the basic system:

1) the shared object hierarchy (i.e., the *dbclass* metaclass),

2) WYSIWYG editors for tools, frames, and forms, and

3) an event-driven scheduler for running several tools in one Common Lisp process.

In addition, Object FADS will be used to develop a series of applications for automating an IC fabrication laboratory. The applications include an editor for process-flow recipes and a work-in-progress (WIP) system for tracking wafers as they move through the fabrication process [HoR87]. These applications will provide a real-world test of the system that we are building.

The first subtask that must be finished is to build the "object cache" that will support the shared object hierarchy. We have already begun the implementation of a *dbclass* metaclass in CLOS. The initial goal is to write a single-user "object cache" that stores data in POSTGRES. This system should be completed by September 1987. It will be used to implement the database representation of Object FADS applications. A multi-user cache that uses the precomputation and distributed cache mechanisms described elsewhere [Row86] will be implemented when the required features are provided in POSTGRES (sometime in early 1988).

The second subtask is to complete WYSIWYG editors for tools, frames, and forms. These editors are essentially Object FADS tools so they will require us to implement most of the features of the system. For example, many different form-types will have to be defined to support these editors (e.g., scrollable text forms, table forms, and graphics forms). Initial versions of these editors should be completed by September 1987.

Lastly, an event-driven scheduler must be developed so that one Common Lisp process can execute several tools concurrently. Common Lisp implementations are typically large programs (e.g., Franz Extended Common Lisp on a Sun workstation is initialized with an 8 megabyte virtual address space and DEC VAX Common Lisp is initialized with a 6 megabyte virtual address space) so it is impractical to run one Common Lisp process per Object FADS tool. We have implemented a prototype event-driven scheduler that will be integrated into the system [Bra86].

We plan to release the first version of Object FADS to users outside Berkeley sometime in the latter part of 1987. We could release a version earlier, but a major, incompatible change is being made to the "X Window System" this summer (i.e., the version 11 protocol) that will require extensive modification of our code.

References

[Boe86] D. B. Bobrow and et.al., "COMMONLOOPS: Merging Lisp and Object-Oriented Programming", *Proc. 1986 ACM OOPSLA Conf.*, Portland, OR, Sep. 1986, 17-29.

[Bra86] R. Brand, *A Portable Multiprogramming Scheduler for Common LISP*, MS Report, Computer Science Division - EECS, U.C. Berkeley, Dec. 1986.

[Get86] J. Gettys, "Problems Implementing Window Systems in UNIX", *Proc. Winter USENIX Technical Conf.*, Jan. 1986, 89-97.

[Gee86] J. Gettys and et.al., *Xlib - C Language X Interface (Protocol Version 10)*, MIT Project Athena, Nov. 1986.

[HoR87] D. A. Hodges and L. A. Rowe, "Information Management for CIM", *Proc. Adv. Res. in VLSI*, Palo Alto, CA, Mar. 1987.

[Mar87] D. C. Martin, *XCL - Common LISP X Interface (Protocol Version 10)*, Computer Science Division - EECS, U.C. Berkeley, Apr. 1987.

[RoS82] L. A. Rowe and K. A. Shoens, "FADS - A Forms Application Development System", *Proc. 1982 ACM-SIGMOD Int. Conf. on the Mgt of Data*, June 1982.

[Row85] L. A. Rowe, "*Fill-in-the-Form* Programming", *Proc. 11th Int. Conf. on Very Large Data Bases*, Aug. 1985.

[Row86] L. A. Rowe, "A Shared Object Hierarchy", *Proc. Int. Wkshp on Object-Oriented Database Systems* , Asilomar, CA , Sep. 1986.

[RoS87] L. A. Rowe and M. R. Stonebraker, "The POSTGRES Data Model", Submitted for publication, Mar. 1987.

[StR84] M. R. Stonebraker and L. A. Rowe, "Database Portals: A New Application Program Interface", *Proc. 10th Int. Conf. on Very Large Data Bases*, Aug. 1984.

[StR86] M. R. Stonebraker and L. A. Rowe, "The Design of POSTGRES", *Proc. 1986 ACM-SIGMOD Int. Conf. on the Mgt. of Data*, June 1986.

[SHH87] M. R. Stonebraker, E. Hanson and C. H. Hong, "The Design of the POSTGRES Rules System", *IEEE Conference on Data Engineering*, Los Angeles, CA, Feb. 1987.

[RTI84] *INGRES ABF (Applications By Forms) User's Guide*, Version 3.0, VAX/VMS, Relational Technology, Inc., Berkeley, CA, May 1984.