Building a User-Interface for the O-Raid Database System using the Suite System*

Teresa L. Nixon Kara Lubenow Jagannathan Srinivasan
Prasun Dewan Bharat Bhargava
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

O-UI, the graphical interface to the O-Raid distributed object-oriented database system, supports access and direct manipulation of relations and objects. Since O-Raid supports a hybrid object-relation data model, an evolutionary approach to design was required. The use of Suite user interface construction tool led to rapid prototyping of the interface, which allowed us to experiment with the interface itself, leading to modifications and refinements to the initial design. In this paper, we describe how O-UI was engineered using Suite, present our experiences, and discuss the related software integration issues.

1 Introduction

The designers of software systems are often faced with requirements that cannot be met by existing systems, which leads to development of new systems. To some extent this process is natural as current systems cannot be expected to meet the needs of all the future applications. However, building a system from scratch is a time consuming effort that involves manpower and computing resources. An alternate approach is to get the required functionality by integrating already built systems. This approach has many benefits. Specifically,

- *Rapid Prototyping*: The resulting system can possibly be built much faster than starting from scratch.
- *High Reusability*: The resources such as software and technology can be reused.
- *Increased Functionality*: The resulting system can not only meet the desired requirements but in addition can inherit additional features from the systems being integrated.

The task of system integration is difficult for many reasons. Systems may be heterogeneous and can differ in the software, interfaces and data models that they use. The systems may also run on different platforms (varying in operating systems, hardware, etc.). The system support documents generally do not contain information such as low level system design to aid the task of system integration.

Despite these drawbacks, system integration is a viable approach for building systems with greater functionality. Integration of heterogeneous database systems [SL90] is becoming a reality. To help the process of integration many standards are being developed.

A process similar to system integration is engineering a system by using an existing software tool. We have engineered a graphical interface called O-UI for O-Raid database system [DVB89] using the Suite [Dew90a] system, which allows near-automatic construction of user interfaces. The O-UI graphical interface supports display as well as direct manipulation of database relations and objects. In this paper, we discuss how O-UI was built and present our experiences.

An interesting aspect of building O-UI is that both O-Raid and Suite employ some form of integration. Specifically, O-Raid integrates object and relational data models [BDMS90], allowing relational and object-oriented applications to coexist. Suite integrates active objects with conventional operating systems allowing Suite objects to coexist with the components of the operating systems [Dew90a]. These components can access and be accessed by Suite objects. However, since O-Raid and Suite are independently developed software, the implementation of O-UI intro-

*This research is supported in part by a grant from AIRMICS, a National Science Foundation grant, and by a grant from General Electric.

0-8186-2697-6/92 $03.00 © 1992 IEEE

duced some redundancies and incompatibilities. The related software integration issues are discussed later in the paper.

The O-UI graphical interface requirements were more complex than those for traditional relational database systems, since O-Raid supports a hybrid object-relation data model. The interface was required to support the display and manipulation of O-Raid relations, where the relation attributes could be arbitrarily complex objects of user-defined types (classes).

Earlier, we had built a simple teletype interface for O-Raid called S-UI, which accepted a query typed by the user and displayed the query result (if any) in a tabular form. The interface was simple and portable but had several limitations. The display of relations with a large number of tuples or attributes was difficult to view. The tabular display of data was awkward for relations containing objects. A flexible mechanism for display of objects was desired. We wanted to display objects through a special *display* method defined for that class, which specifies how the objects of that class should be displayed[AGS90]. Another limitation was that the query result could not be reused for a subsequent query. This increased the effort required to obtain desired information through a series of steps (query refinement). Also, the manipulation of relations could only be done by specifying update queries.

To overcome these limitations we embarked on building the O-UI graphical interface. We wanted to build the graphical interface based on the direct manipulation paradigm [Shn83], with features such as mouse based interactions, pop-up menus, windows, icons, and graphical display of data. Another goal was to minimize the amount of information the user has to know (such as query language syntax, etc.) and reduce the data that needs to be typed.

We considered using X windows and the associated toolkits (such as [MA88]) for building the interface. Toolkits implement a set of user interface features such as menus or command buttons, commonly referred to as toolkit *widgets*. The widgets can be manipulated using object-oriented paradigm. X windows also has an additional layer, Xt Intrinsics, which allows the user to create new widgets. The implementation that used X and the associated toolkits required a lot of learning and involved major programming effort. Also, the implementation was not easy to change.

Next, we looked into two tools that ease the task of building user interfaces: Interviews [LVC89] and Suite [Dew90a]. We chose Suite over Interviews as Suite provides a dialogue manager to manage the user interface aspects of the application. In contrast, using Interviews still required us to build and manage the interface. The Suite dialogue manager supports direct editing of arbitrarily complex data structures. The direct editing capability was suitable for supporting direct manipulation of O-Raid relations. Also, the dialogue manager allows flexible display of data where the display can be customized at run-time by the user. This feature allowed us to experiment with the interface and choose a suitable scheme for displaying the query results.

In addition, Suite is a good choice for the following reasons:

- It automated the construction of most parts of the user interface. The system itself is built on top of X windows, and we did not have to understand the details of the X layer.
- It allowed rapid prototyping of the user interface, which was essential for the evolutionary approach adopted in the design of O-UI. For the unconventional object-relational data model we had no prior experience or understanding of what would be a good set of features. The best policy was to build a prototype, experiment with it, and refine the design.
- The software was developed on the same platform (a network of workstations supporting UNIX, TCP/IP and X) as O-Raid.

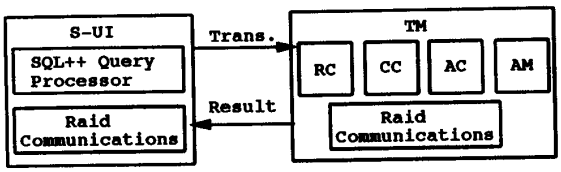
The rest of the paper is organized as follows. Section 2 gives an overview of O-Raid and Suite. Section 3 presents a detailed design specification of the graphical interface. Section 4 describes how O-UI was implemented using Suite. Section 5 presents our experiences and discusses the related software integration issues. Finally, we outline the future research plan and give conclusions.

2 Overview of O-Raid and Suite

In this section, we briefly describe the two systems. We also discuss how Suite can be used to display and edit data structures of an interactive application.

2.1 O-Raid Overview

O-Raid is an extension of RAID [BR89] distributed relational database system that provides support for objects [DVB89, BDMS90]. The key features of O-Raid are:



LEGEND

RC = Replication Controller

CC = Concurrency Controller

AC = Atomicity Controller

AM = Access Manager

Figure 1: O-Raid S-UI and TM

- support of relations whose attributes can be of user-defined types (classes)
- use of C++ programming language [Str86] for defining classes
- use of SQL++ query language (an extension of SQL), which includes constructs to allow subobject and method referencing [MSDB90].

Figure 1 shows the interaction between the existing simple teletype interface S-UI and O-Raid Transaction Manager (TM). The TM itself consists of four servers namely Replication Controller (RC), Concurrency Controller (CC), Atomicity Controller (AC), and Access Manager(AM). The user submits a SQL++ query by typing the complete query. The S-UI parses the query and generates a transaction of read and write operations on relations. The transaction is submitted to TM, which processes the transaction and returns the result (if any) to S-UI. The S-UI displays the query result on the screen. The communication between TM and S-UI is done through UDP messages [MB91].

2.2 Suite Overview

Suite provides support for editing data structures described by the data types of conventional programming languages. The user interface management is provided by a dialogue manager (DM). It displays selected data structures of applications providing the user with a generic editor-based user interface to modify these data structures in a syntactically and semantically correct fashion. DM also allows the applications to trigger semantic actions (such as displaying the results and communicating with other applications) in response to user changes to the data structures. Applications specify the display properties of the data structures, and a multiple inheritance scheme [Dew91] is provided for specifying default values of these properties.

Figure 2 shows how an interactive application is supported in Suite. The interactive application is partitioned into a dialogue manager which manages the user interface of the application and an editable (active) object which defines the semantics. The communication between the dialogue manager and the editable object is done using remote procedure calls (RPC). When a data structure is updated through the dialogue manager, a corresponding update procedure defined in editable object is invoked through the RPC mechanism. The editable object in turn can invoke the Update_DM procedure defined in dialogue manager to update the display of some data structure.

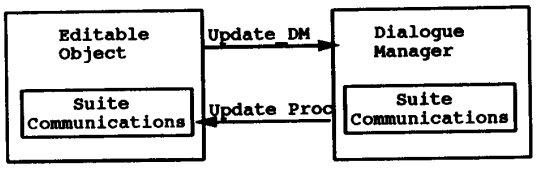


Figure 2: A Suite Application

3 O-UI User Interface Architecture

Based on the requirements of O-Raid, we did a detailed design of O-UI. Here we present the design specification.

The user interface has a main window, called the query-interface window, and one or more additional windows, called relation-display windows (see Figure 3).

3.1 The query-interface window

The query-interface window is used to display project information and to enter the SQL++ Queries. Its features include:

- SQL++ Query Line: A standard SQL++ query can be entered on this line, and it will be sent (as is) for further processing.
- SQL++ Query Status Line: When a query has been processed, a status message will be printed as to whether or not the query was successful and if results were returned.

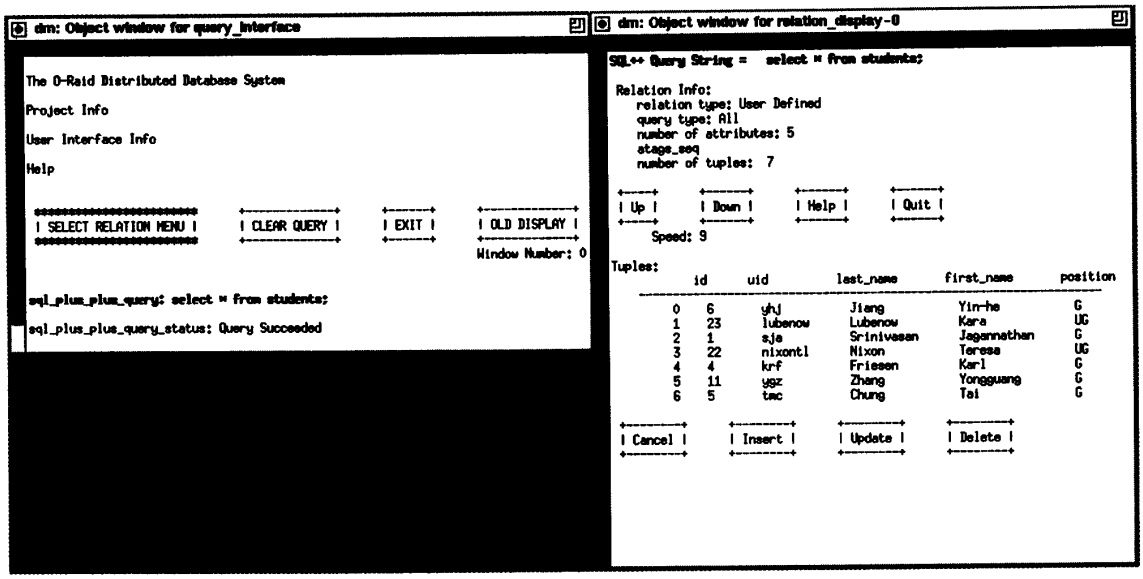


Figure 3: Query Interface and Relation Display Window

- Clear Query Button: This button is used to clear the SQL++ Query Line and the Status Line, so the user does not have to back space through the previous query.
- Display Button: With this button, the user decides which relation-display window the output of the query must be sent. Whenever a query is done, a new relation-display window is created to display the results. If the user wants the output to go to an old relation-display window instead, this button is selected and the user chooses one of the old relation-display windows by specifying the corresponding window-id.
- Select Relation Menu Button: This menu displays all the system relations. When the user selects one of the relations, a select command (SQL++ query to display the entire relation) is generated, and the corresponding query is processed. The menu also contains an *other* field which generates part of the select query and lets the user specify the desired relation. The *other* option is useful for viewing user defined relations.
- Exit Button: The Exit button exits from the query-interface window and removes all the associated relation-display windows.
- Information Fields: The fields Project Info, User Interface Info, and Help can be selected to get information about the system.

3.2 The relation-display windows

The relation-display windows (shown in Figure 3) are used to display the relations and information about their attributes and tuples such as number, type, and length. The user is able to edit the manner in which the display is shown and also edit the actual relation within the system.

Editing the display: Initially the relation is displayed horizontally with headers on top of each attribute column. The system attributes are not shown. Some of the features available for editing are:

- Eliding columns: By clicking on the header of a column, the user is able to elide the entire column.
- Up and Down Buttons: These buttons support scrolling of relations, which do not fit in a normal screen. For those relations, only a part of the relation is displayed. By clicking on either the Up or Down Button, the user can scroll the relation.
- Help Button: On clicking this button help information describing the options available to the user is presented.

- Quit Button: This button is used to remove a relation-display window after the user has finished viewing the result.

Editing the actual relation: To minimize the amount of typing and information the user must know, the user interface allows the relation to be edited by changes made to the display. The options available are:

- Delete Button: The delete option creates a query to delete the highlighted tuple, sends the query to the O-Raid transaction manager and displays the modified relation.
- Insert Button: The insert option allows the user to enter the new tuple values in an uninitialized tuple displayed at the bottom of the window. After the tuple values are entered, an insert query is created and sent to the O-Raid transaction manager to be processed. The modified relation is then displayed in the window.
- Update Button: The update option copies the highlighted tuple to the bottom of the window (see Figure 4). After the tuple is edited, an update query is created and processed by the O-Raid transaction manager. The changes are then reflected in the displayed relation.

When a select query is done in the query-interface window with specifications as to which attributes will be shown, a temporary relation-display window is created. Temporary windows allow the user to edit the display but they do not have the options to edit the actual relation.

4 Implementing O-UI using Suite

O-UI was built in two phases. Initially, a feasibility study was done to uncover any incompatibility problems in using Suite. Next, a detailed implementation effort was undertaken to support all the features presented in the previous section.

4.1 Phase I: Feasibility Study

A Suite application called **query_interface** (QI) was developed, which accepts an SQL++ query, parses it, and generates an equivalent transaction. This transaction is submitted to O-Raid TM for further processing. The result returned by TM is then displayed (see Figure 5). Suite mechanisms were used

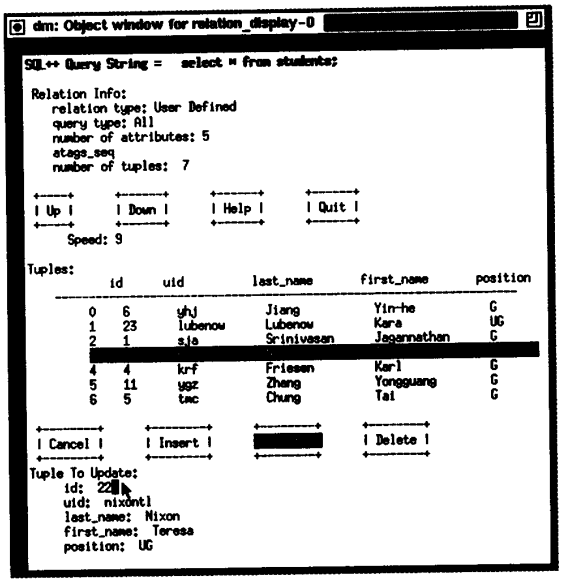
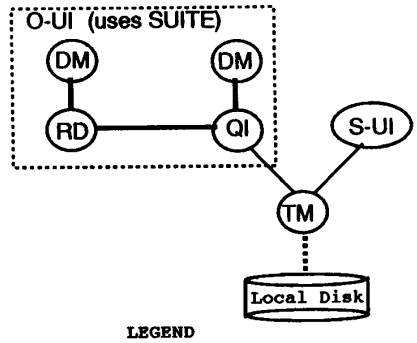


Figure 4: Updating a tuple of a relation

for displaying as well as editing the following two data structures of this application:

1. *SQL++ Query String:* This data structure is used to read in an SQL++ query. The Suite dialogue manager associated with the **query_interface** application is used to edit this data structure as a means of submitting an SQL++ query. The application sets up a trigger which is activated whenever this data structure is edited. The trigger results in the parsing and execution of the submitted SQL++ query.
2. *SQL++ Query Result:* This data structure is used to display the results whenever a query is executed. The query result is a relation (table). Again, the associated Suite dialogue is used to view as well as to customize the display of results.

We were able to build the basic infrastructure of QI in a month's time. The task was made easier by the existing simple teletype query interface (S-UI) to O-Raid. S-UI, written in C using yacc [Joh] and lex [LS] tools, was reorganized by packaging all SQL++ query processing routines into a single library. This library is now shared by both S-UI and QI. QI includes Suite specific commands embedded in C comment statements that specify the displayable and editable properties of the two data structures mentioned above. QI is run through a Suite preprocessor to generate the



LEGEND

TM	=	Transaction Manager
QI	=	Query-Interface
RD	=	Relation-Display
DM	=	Dialogue Manager
S-UI	=	SQL++ User Interface
O-UI	=	O-Raid Graphical Interface
▬▬		Suite RPC
——		RAID Communications
·····		Disk access

Figure 5: O-Raid Process Layout

corresponding C programs. The resulting programs are then compiled using a standard C complier (for more details about Suite applications see [Dew90b]).

4.2 Phase II: Implementation

The success of the feasability study led us to undertake the implementation of entire O-UI using Suite.

O-UI Organization: Figure 5 shows the current configuration of O-UI. The two Suite applications, namely, **query_interface** (QI) and **relation_display** (RD), correspond to the proposed query-interface and relation-display windows. The user interacts with each of these applications through the associated dialogue manager (DM). The user submits a query to QI through the associated dialogue manager. The query is sent to the O-Raid Transaction Manager (TM). The TM processes the query and returns the result to QI. QI uses the Suite RPC mechanism to send the result to RD. The dialogue manager associated with RD displays the result. The query result displayed can be further customized. In addition, when user defined relations are displayed, they can

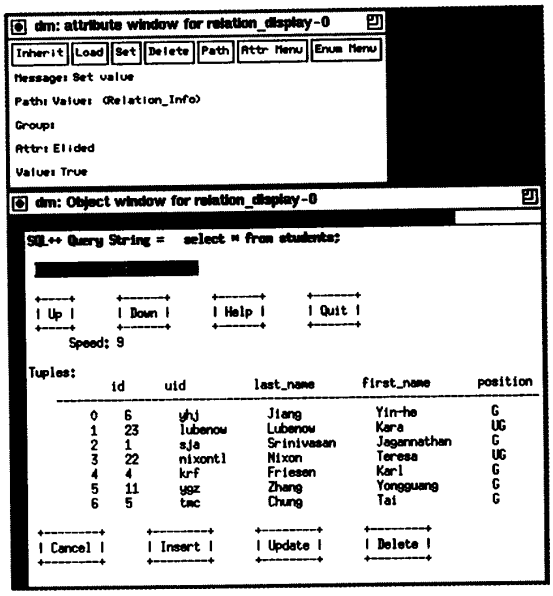


Figure 6: Eliding a field using the dialogue manager attribute window

be directly manipulated through the dialogue manger. Figure 5 also shows how the simple teletype interface S-UI interacts with the O-Raid Transaction Manager.

Supporting flexible display of query results: The dialogue manager provides options for customizing the display of data structures. The customization can be done either at run time or it can be done in the Suite application itself through calls to dialogue manager routines. We provide a default customization where the tuples of the relation are displayed horizontally as in Figure 3. The run-time customization is done by using the dialogue manager *attribute* window. For example, to hide the attribute *Relation_Info* the user can open the attribute window and set the elided attribute to true as shown in Figure 6. For large relations, only portions of relation are shown and scrolling of relation is accomplished by displaying subsequent sets of tuples of the relation.

Supporting direct editing of relations: We made use of dialogue manager mechanisms for editing the displayed relation. For example, to update a tuple of the relation, the user selects the desired tuple and makes changes (as shown in Figure 4). Suite recognizes the part of the data structure selected (tuple in our case). After the updates are performed, an

equivalent update query is generated. The generated query is sent to QI through a remote procedure call. QI sends the query to TM. A select query for the relation that was recently updated is sent and the result is now displayed on the relation-display window. Also, any other window that has data pertaining to the same relation is automatically updated to reflect the latest changes. In similar manner, insert and delete operations are implemented.

Simulating push buttons: The O-UI design required push buttons for both query-interface and relation-display windows. Suite only allowed user defined buttons on the dialogue manager menu window. It did not allow adding buttons on object windows (where the data is displayed). The buttons were simulated in the following manner. Suite allows associating a selection procedure with a variable. The selection procedure is called whenever the variable is selected. The push buttons were implemented as variable fields with an associated selection procedure. When the variable field is selected the associated selection procedure gets called, which in effect simulates a push button.

Simulating pop-up menus: Suite did not support user-defined menus. However, it creates a menu for editing the variables of enumerated data type. This mechanism is used for creating menus. The only restriction is that menu options had to be statically determined. For example, the *select relation menu* is implemented by defining a variable of enumerated data type. When the variable is edited, the associated update procedure gets called, which generates the appropriate query based on the menu option selected.

Using the mechanisms discussed above we were able to build the complete O-UI interface. QI also contains code to retrieve the metadata information about the O-Raid database. The information is displayed to the user on request.

5 Experiences

Overall, Suite was appropriate for building O-UI. Here we discuss the benefits as well as the difficulties encountered in using Suite. Since O-Raid and Suite are two independently developed systems, the engineering of O-UI led to some incompatibilities and redundancies. We also discuss the related software integration issues.

5.1 Suite as a user interface construction tool

Using Suite we were able to build the O-UI in about six months (see Table 1). The following features helped us in rapid prototyping the system:

- *Flexible display of data*: Suite dialogue manager provides flexible display of data (in our case relations). We were able to customize the relation display at run-time using the dialogue manager options. It helped us in experimenting and choosing a suitable display for query results. In addition, the user has the flexibility of further customizing display of results according to her taste and needs.

- *Direct manipulation of data*: We were able to directly manipulate the displayed relations through the Suite dialogue manager. For example, supporting updates on relations was straight forward. Suite provided some of the needed features such as determining the parts of data structure selected and triggering appropriate action routines.

- *Ease of building the user interface as a collection of Suite applications*: Usually a user interface consists of multiple windows. Suite allowed us to map different windows to separate applications. Since Suite provides a remote procedure call mechanism (RPC), the task of collectively meeting user interface requirements was easy to support through a collection of Suite applications, where these applications communicate with each other through RPC.

Although Suite greatly reduced the task of building the user interface, we did face some difficulties. Specifically,

- *Restricted support for push buttons and menus*: Push buttons and menus occur quite commonly in a user interface. Suite only allowed user defined push buttons to be added to dialogue manager menu window. We required the buttons to be placed in the window where the data is displayed. We had to simulate the push buttons. Moreover, the looks of the buttons where not satisfactory. We faced a similar problem with menus. Suite provided menus as a means of editing a variable of an enumerated data type. We had to simulate menus using this mechanism. This required us to fix the menu items at compile time.

- *Interaction through dialogue manger causes extra overheads*: Since the user interface aspects of the application is managed by the dialogue manager, the interaction with the interface is always through the dialogue manager. Thus we require two processes per application, the suite application and the associated dialogue manager, as opposed to a single process. Also, there is extra communication overhead as every event (such as editing the data structure) is first received by the dialogue manager, which then performs a remote procedure call to invoke the appropriate routine of the corresponding application.

5.2 Software Integration Issues

The building of O-UI resulted in duplication of software. The current O-UI uses two communication libraries: Suite RPC to communicate between Suite applications (including the dialogue manager) and the Raid communication library to communicate between QI and TM (see Figure 5). Similarly both Suite and O-Raid have separate mechanism for maintaining persistent data. These mechanisms are incompatible. That is, persistent data stored using Suite can not be accessed directly using O-Raid and vice versa.

The problems of redundant software and incompatibilities are bound to occur when independently developed software are integrated. The term post-facto integration has been used to refer to such integration of already existing software [Pow90]. The integration process benefits from the reuse of software leading to quick prototyping.

An interesting alternative is to consider modifying the software components (in our case O-Raid and Suite) to avoid redundancies and incompatibilities. Suite offers excellent display and direct manipulation capabilities, whereas, O-Raid supports efficient storage and manipulation of persistent data. One possibility is to modify Suite to make use of O-Raid mechanism for supporting persistent data. On the other hand the two communication mechanisms, Suite RPC and Raid communications, should be retained as they increases the reusability of the integrated software.

5.3 Engineering Efforts

Table 1 summarizes the engineering effort. We (two of the authors working half-time) were able to build the initial version of the graphical interface in less than a month, where we tested the feasibility of using Suite. We experimented with the O-UI prototype and did a detailed design. The proposed features were were then implemented by a student programmer working (full-time) during the summer in about three months. The three month period includes the time needed for getting familiar with the Suite system. The current version supports all the features discussed here. The O-UI consists of two Suite applications, `query_interface` and `relation_display`, each about 1000 lines long (None of the code deals with X).

Task	Time (in months)	Persons involved
Learning Suite and the Feasibility Study	One	Two (half-time)
Detailed Design of O-UI	One	Two (half-time)
Implementing O-UI using Suite	Three	One (full-time)

Table 1: Engineering Efforts

6 Conclusions and Future Work

Several factors contributed to the success of this engineering effort. The rapid prototyping capability offered by Suite was suitable for the evolutionary approach adopted in designing the graphical interface. Our design is expected to evolve further (for example to graphical display objects stored in relations, invoke methods on objects, etc.).

The existence of a simple teletype interface (S-UI) for O-Raid aided us in building the O-UI graphical interface. Specifically, the interface between S-UI and the O-Raid Transaction Manager was retained in O-UI. S-UI software was reorganized by packaging all the the query processing routines into a single library. This library is now shared by both the simple and the graphical interface.

The engineering task becomes much easier if the systems being used adhere to standards. This was the case as both Suite and O-Raid run on a network of workstations executing UNIX, TCP/IP, NFS and X. Furthermore, the task of writing Suite applications involved little learning as it used the popular C programming language.

Using a system like Suite opens the potential for interesting features and experimentation. For example, Suite supports flexible coupling among shared windows of a user interface [DC91]. Thus we can ex-

periment with mechanisms of sharing and updating of windows of multiple users.

Suite, though built using X and associated toolkits, did not provided features such as as push buttons and pop-up menus. We had to simulate these features. One possibility is to use Suite till the design process has stabilized and then to implement the entire interface using standard tools such as X and its associated toolkits.

Although the reuse of software is desirable, it can be difficult as one has to learn and understand the software being used. In our case, the designers and implementers of the interface had no prior experience or knowledge of Suite. The task of learning the Suite system was done mainly through trying out existing Suite applications. The learning was much easier as Suite applications were fairly small (usually about 250-500 lines) and were written in C.[1] We interacted with the designers of Suite and took advantage of, as well as suggest new features. In general, the designers of a software tool can not anticipate all the possible ways the tool would be used. However, close interaction is not always possible and in such cases good document support with examples showing different possible uses of the tool should be provided.

In the future we plan to provide support for viewing as well as manipulating objects stored in O-Raid relations. Specifically, we plan to build support for: creating and inserting tuples containing objects, invoking methods on O-Raid objects, displaying objects graphically, and following the pointers to other objects.

A simple display scheme for objects is to show all the *public* attributes of the object in a tabular form. In addition, we plan to support a flexible display of objects by defining *display* method(s) for each of the classes and using them to construct the display. The *display* method specifies how the object should be displayed as in [AGS90].

The objects displayed can be updated directly or through the methods defined for the corresponding class. The updates through methods will be supported by providing user with a menu of possible methods. Once the user selects a method, a template will be displayed requiring the user to fill in the argument values for the method. After the arguments have been filled in, an equivalent update query would be submitted for performing the update on the database. The required metadata information will be loaded into the Suite application by performing queries on O-Raid system relations. The information about a class is added to the O-Raid system relations when the class is registered with the database [BCJS].

Several research questions need to be resolved when we consider performing direct updates on the objects, that is, through editing the display of object. Objects are accessed and manipulated through the methods defined for the corresponding class. Should we support direct updates on objects? One possibility is to allow direct updates on *public* attributes of the objects. How about if these attributes have dependencies on other attributes of the object? We are considering these issues in designing support for direct manipulation of O-Raid objects.

[1] Suite applications tend to be small as most of the user interface related code is automatically generated by Suite preprocessors.

7 Acknowledgements

Rajiv Choudhary helped us in debugging our code and explained some of the advanced features of Suite. We thank the anonymous referees for their comments on a previous draft which helped us in revising the paper.

References

[AGS90] R. Agarwal, N. H. Gehani, and J. Srinivasan. Odeview: The Graphical Interface to Ode. In *Proc. ACM-SIGMOD Conf. on Mangement of Data*, Atlantic City, New Jersey, May 1990.

[BCJS] B. Bhargava, T. M. Chung, Y. Jiang, and J. Srinivasan. Design and Implementation of the O-Raid Data Definition Facility. Submitted for publication.

[BDMS90] B. Bhargava, P. Dewan, J. G. Mullen, and J. Srinivasan. Implementing Object Support in the RAID Distributed Database System. In *Proceedings Of The First International Conference on Systems Integration*, pages 368–377, April 1990.

[BR89] B. Bhargava and J. Riedl. The RAID Distributed Database System. *IEEE Transactions on Software Engineering*, 15(6), June 1989. (New version report in preparation).

[DC91] P. Dewan and R. Choudhary. Flexible User Interface Coupling in Collaborative Systems. In *Proceedings of the ACM CHI'91*

Conference, pages 41–49. ACM, New York, 1991.

[Dew90a] P. Dewan. A Tour of the Suite User Interface Software. In *Proceedings of the 3rd ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 57–65, October 1990.

[Dew90b] P. Dewan. A Guide to Suite. Technical Report SERC-TR-60-P, Software Engineering Research Center, Purdue University, West Lafayette, Indiana, February 1990.

[Dew91] P. Dewan. An Inheritance Model for Supporting Flexible Displays of Data Structures. *Software-Practice and Experience*, 21:719–738, July 1991.

[DVB89] P. Dewan, A. Vikram, and B. Bhargava. Engineering the Object-Relation Model in O-Raid. In *Proceedings Of the International Conference on Foundations of Data Organization and Algorithms*, pages 389–403, June 1989.

[Joh] S. C. Johnson. YACC - Yet Another Compiler Compiler.

[LS] M. E. Lesk and E. Schmidt. LEX - Lexical Analyzer Generator.

[LVC89] M. A. Linton, J. M. Vlissides, and P. R. Calder. Composing User Interfaces with InterViews. *IEEE Computer*, pages 8–24, February 1989.

[MA88] J. McCormack and P. Asente. An Overview of the X Toolkit. In *ACM SIGGRAPH Symposium on User Interface Software*, October 1988.

[MB91] E. Mafla and B. Bhargava. Communication Facilities for Distributed Transaction-Processing Systems. *IEEE Computer*, 24(8):61–66, 1991.

[MSDB90] J. G. Mullen, J. Srinivasan, P. Dewan, and B. Bhargava. Supporting Queries in the O-Raid Object-Oriented Database System. In *Proceedings of the International Computer Software and Applications Conference*, 1990.

[Pow90] L. R. Power. Post-Facto Integration Technology: New Discipline for an Old Practice. In *Proceedings Of The First International Conference on Systems Integration*, pages 4–13, April 1990.

[Shn83] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16:57–69, 1983.

[SL90] A. Sheth and J. L. Larson. Federated Databases: Architectures and Integration. *Computing surveys*, 22(3), September 1990.

[Str86] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, Mass., 1986.