User Interface Reengineering

Ph.D. Research Proposal
Update Summary

Melody M. Moore

College of Computing
Software Research Center / Open Systems Laboratory and
Graphics, Visualization, and Usability Center
Georgia Institute of Technology

Feb 19, 1996

Committee: Dr. James Foley (chair)
Dr. Gregory Abowd
Dean Peter Freeman
Dr. Richard LeBlanc
Dr. Ettore Merlo (Ecole Polytechnique de Montreal)
Dr. Spencer Rugaber

User Interface Reengineering

Proposal Update Summary

Purpose

This document is an update to a research proposal originally submitted in June 1994 [MOO94a]. Since that time, research in this field has progressed, necessitating changes and additions to the initial proposal. While the majority of the original proposal is still valid, this addendum summarizes the evolution of the research and plans.

The Reengineering Problem

The increasing need to migrate systems across very different platforms has created difficult problems for software maintainers. Often, it is simpler to completely rewrite the old application rather than attempting to adapt it to the new environment. However, rewriting large systems is prohibitive in both time and cost. Transition technologies, which allow applications to be quickly migrated to new environments with automated assistance, are becoming critical to the maintenance process. Although reengineering research has progressed substantially, the focus has been on techniques and tools to recover algorithms and data models from legacy systems. However, these tools do not address a significant and difficult problem in migration - reengineering the user interface. With estimates asserting that up to 50% of the code of an interactive system is devoted to the user interface [SUT78], this is a significant deficiency in maintenance technology. Furthermore, the user interface is the most likely component of the system to change drastically during migration. Often the entire legacy system is thrown away and rebuilt from scratch because of the difficulty of migrating the user interface.

Many of these aging information systems were originally implemented with character or text-based user interfaces, and are usually menu-driven. Migration to newer platforms, such as graphical workstations, requires the text-based user interface to be replaced by graphical interaction techniques. The default interaction style of almost all modern interactive systems is “WIMP” (Windows, Icons, Menus, and Pointers) [DIX93]. Since there is a dramatic paradigm shift between the control aspects of a text-based interface and a WIMP interface, it is not sufficient to simply replace the textual output statements of the legacy system with graphical counterparts. The user interface components and their interaction with the computational components of the system need to be understood in order to restructure the application. Traditional reverse engineering techniques have not addressed recovering user interface models from code, a necessary step in automation-assisted migration.

Revised Thesis

Adding automation to the reengineering process can significantly reduce the time, cost, and effort of migrating systems. It is possible to apply reverse engineering techniques to text-based legacy applications in order to recover and extract an abstract model of the user interface. The resulting model can then be transformed to generate a new graphical WIMP-style user interface for the system.

A Process for Reengineering

There are three steps in the reengineering process:

- *Detection* - (also called Reverse Engineering) Analyzing source code to identify and extract user interface components from the legacy system.
- *Representation* - Building a model of the existing user interface from the detection step.
- *Transformation* - Manipulating, augmenting, or restructuring the resulting model to allow forward engineering to a graphical environment.

Figure 1 describes options in the reengineering process:

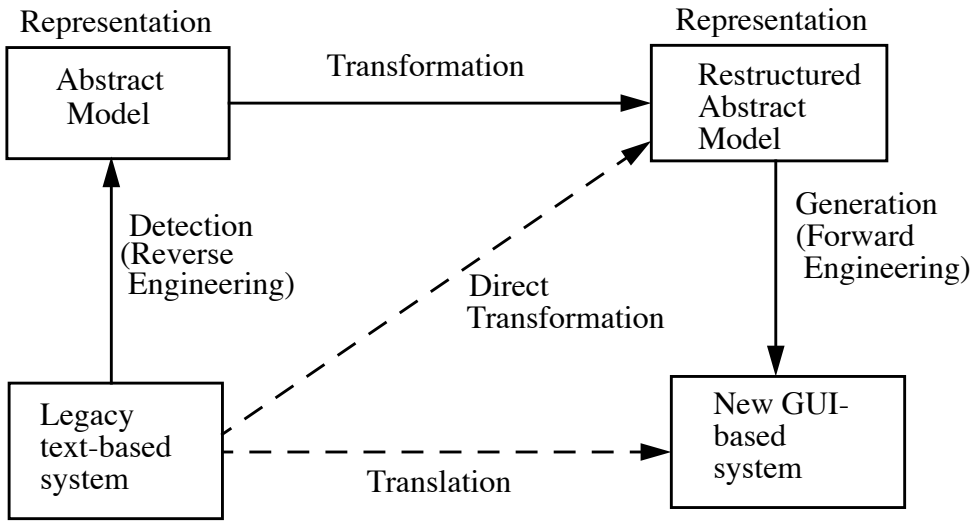

Figure 1 : The Reengineering Process

There are several options for reengineering user interfaces in this process model. The user interface calls in the text-based legacy system could be simply replaced by GUI counterparts, represented by the "translation" edge in Figure 1. There are tools currently available ("screen scrapers") that accomplish this. However, this strategy does not address the necessary shift in control from application-driven to interface-driven (discussed in more detail in the "Modeling Strategy" section below). A better strategy is to extract the user interface from the computational legacy code, using program understanding techniques to build an abstraction, or model, of the existing interface. This path is depicted by the "detection" edge and the "abstract model" box in Figure 1. The application-driven model can then be transformed to an interface-driven model, shown by the "transformation" edge. Once the model has been restructured, a forward-engineering tool, such as a UIMS, could be used to automatically generate a new graphical interface for the system. Another possibility is that the interface-driven model could be directly detected from the legacy code, without modeling the existing structure, shown by the "direct transformation" edge. This research will address the detection-transformation path shown in the diagram.

A Modeling Strategy

Text-based legacy applications are almost always *Computation-Dominant* as opposed to *Dialog-Dominant*. [HAR89]. This means that the user interface is intertwined in and controlled by the computational algorithms of the application. Most modern GUI interfaces are just the opposite; the user interface dialog structure controls the application (Dialog-Dominant). In order to transition the Computation-Dominant legacy system to the new Dialog-Dominant paradigm, the legacy system must be restructured.

Green's Seeheim model [GRE85] defines an architecture for an interactive system that separates the computational code from the details of the user interface. The Seeheim model is the basis for many User Interface Management Systems (UIMS), which allow designers to specify the presentation, dialog control, and application interface separately from the computational code. A significant advantage could be gained by reverse engineering an existing text-based system into a representation that could be then transformed into a UIMS representation. Since the legacy application could be maintained separately from its user interface, the user interface could be tailored and updated as needed without having to modify the computational code. Another advantage of separating the user interface from the computational code is in migration to client-server architectures. Reverse engineering to a Seeheim model requires extracting several different models from the original legacy code, as depicted in Figure 2:

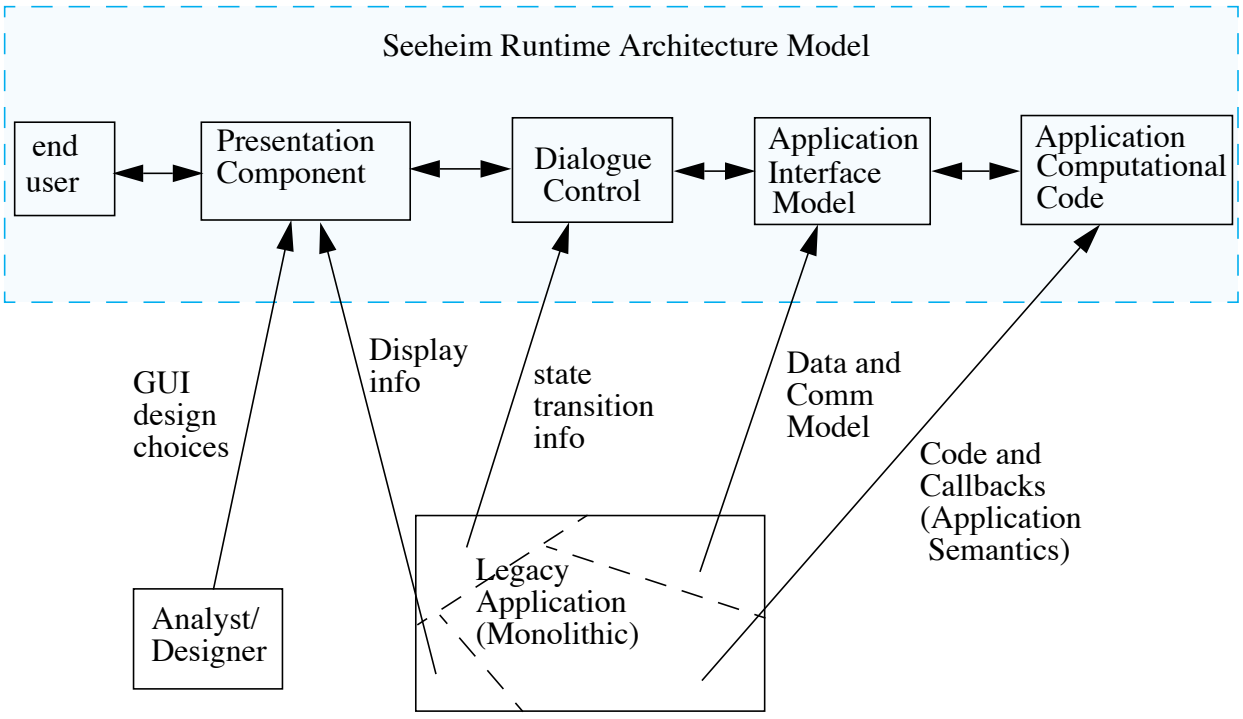

Figure 2 : Reverse Engineering to the Seeheim Model

- The *Presentation Component* of the legacy system includes information such as output groupings (such as in an error message), output-input pairs (such as in a dialog box), contents of screens, and any visible state information. Since the presentation changes drastically between a text-based system and a GUI-based system, the analyst/designer may need to supply further presentation information, such as choosing a selection mechanism (pushbuttons vs. a cascade menu, for example).
- *Dialogue Control* is essentially a "map" of the system, its states and transitions, and also storage for information that allows the interface to be traversed. Sequencing of menus and dialogs are kept in this model, and as a result, control flow analysis can be used to build it. Preconditions and Postconditions are also maintained here to ensure control to flow only in allowed circumstances.
- The *Application Interface* is the connection between the application and the user interface, essentially a communication model. Values and messages to be displayed are computed in the algorithmic code, and relationships between these "user interface variables" and corresponding user interface objects must be maintained. Slicing and dataflow analysis can be used to extract this model from the legacy code.
- The *Computational Code* is everything that is left when the user interface models have been extracted. Although the computational aspects of the code should not need to be altered, the control structure of the program is likely to be reorganized to fit the new Dialog-Dominant paradigm. The result is that the computational code may be restructured in the form of callbacks that respond to user input events.

The user interface components that can be extracted directly from the legacy code consist of parts of the Presentation component and the Dialogue control component. The Application Interface is built as a result of the reverse engineering step (although information needed to build this model is derived from reverse engineering).

A Technique for Reengineering

As described in the original proposal and in [MOO94b], I have developed a technique for reverse engineering user interfaces based on a set of recognition rules. An initial model is built using static analysis, which is then refined during dynamic analysis and then transformed into a representation that supports generation of a new user interface. A new facet of this technique is the goal of extracting and building several user interface models and the connections between them from the legacy code to match the components of the Seeheim model. This will allow the user interface to be transformed into a UIMS representation for automated generation of the new user interface. Figure 3 illustrates the steps in the technique:

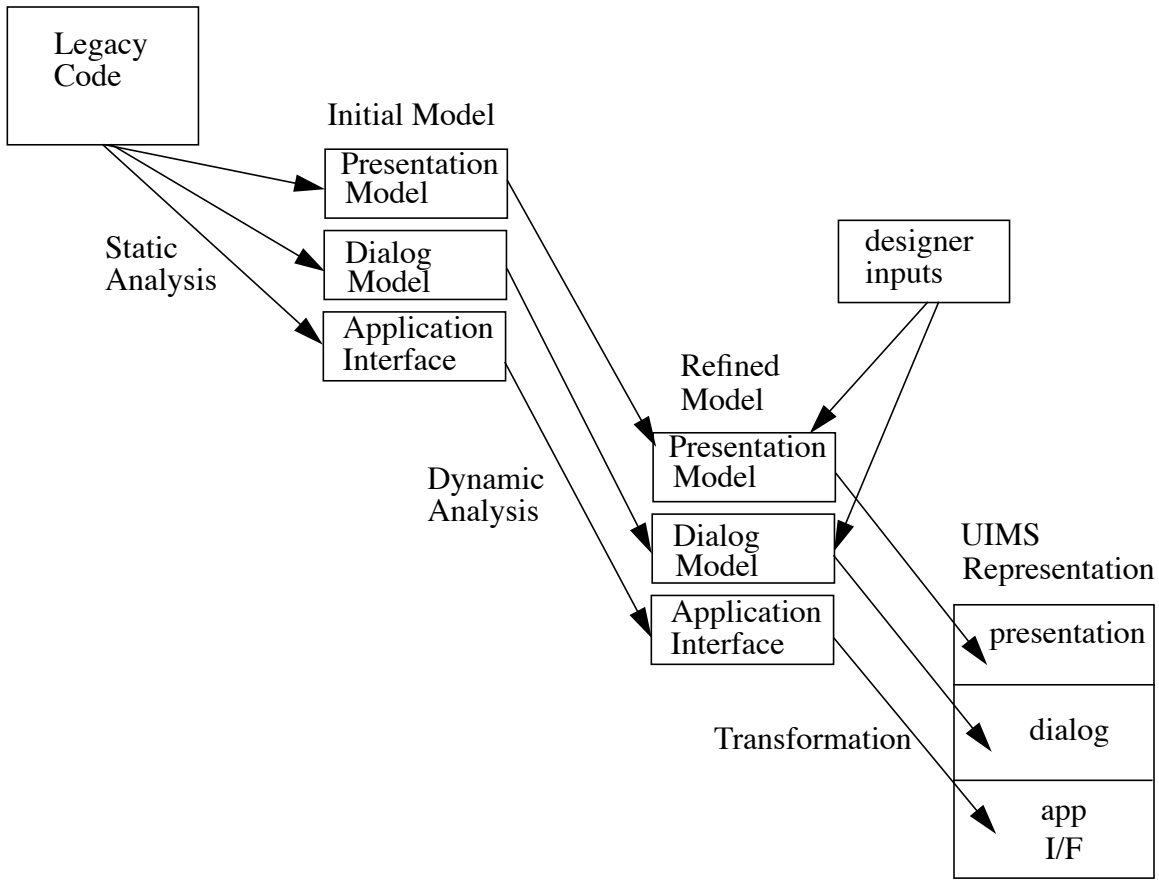

Figure 3 : Technique for Reengineering User Interfaces

Static Analysis

The static analysis step of the reengineering technique begins with the identification of the "User Interface Slice" of the system. The procedural code is translated to an Abstract Syntax Tree (AST) and then, using control flow and data flow analysis, the components of the legacy code that are involved in I/O are identified. The recognition rules are applied statically to the AST to identify user interface paradigms in the code and to build the initial model.

Dynamic Analysis

Static analysis is limited in a number of ways. In order to be able to broaden the scope of applications that could be reengineered and make the technique more general, dynamic analysis allows:

- Resolution of aliasing problems (ie, pointers to functions) that cannot be resolved statically.
- Resolution of filename and other dynamically bound issues
- Terminal control semantics - often text-based systems make use of simple terminal control techniques, which can add semantics to the interface that cannot be detected statically.
- Path analysis and usage data - instrumentation can be used to gather data about often used

sequences and potentially for unused code detection.

- Dynamic analysis could be used in conjunction with static analysis to provide a more complete and accurate model of the interface. Adding static analysis as a first step reduces some of the issues with dynamic analysis, such as path coverage. Comparisons between static and dynamic approaches would be interesting also.

Transformation

In order to shift the control paradigm for the user interface, the legacy system must be restructured to reflect a dialog-controlled model rather than a computation-controlled model. To accomplish this, the dialog control model and parts of the application interface elicited from the detection step needs to be transformed to reflect a more event-driven organization. Action procedures in the computational code need to be identified and restructured in order to serve as callback mechanisms, which is maintained by the application interface. A communication model is also needed to allow data to be passed back and forth from the computational code to the dialog control and presentation components of the model.

Detection of Higher-Level Semantics

Current work has been successful in detecting basic user interface paradigms, such as menu and screen structure, and prompt-response dialogs in purely sequential interfaces. However, there are many capabilities present in GUIs that are not present in text-based interfaces, for example graying out commands that are unavailable or determining which sequences of commands could be performed concurrently (such as multi-threaded dialogues). This work might entail identifying user interface objects from the legacy code and attempting to restructure the code to allow concurrency. Other higher-level semantics might include preconditions and postconditions of commands, and passive status indicators (for modes or toggles).

Recent Research Results - Representation Issues

In order to choose appropriate notations and modeling techniques for each of the three Seeheim model components, I undertook a study of existing user interface representation techniques. After an extensive survey of the literature, I chose ten candidate notations in seven strategic categories, detailed in [MOO96]. In a series of thought experiments, I reengineered a small legacy information system using each of the notations in order to evaluate them for a number of criteria including recoverability, human understanding, expressive power, and capability to express the requirements of the transformed system (ie, preconditions, postconditions, concurrency). The results showed, not surprisingly, that no single representation technique is sufficient to describe all of the user interface models. Consequently, I plan to develop a hybrid representation, based on Harel's State Charts for dialog control, adding a data and connection model for the application interface and a data model for the presentation component model. I plan to pursue an object-oriented data model in order to better reflect the transformation to the WIMP-style interface, and to make it easier to integrate a UIMS as a forward engineering tool.

Background Developments

Since 1994, a very significant achievement in user interface reengineering was accomplished by Merlo et al [MER95] at the Computer Research Institute of Montreal. As part of the Macroscope

project a method, representation, and supporting toolset for migrating a character-based legacy system to a graphical system were developed. A summary of their accomplishments:

- Static analysis of legacy code, including:
  - Parsing to an Abstract Syntax Tree
  - Slicing to isolate user interface components
  - Using a cliche recognizer to produce an initial specification
  - A representation, AUIDL [MER93], based on Milner's process algebra for dialog control
  - An object-oriented data model incorporated into AUIDL
  - Allowing the user to manipulate the model (a graphical representation is used)
  - Generation of the new interface
- Targets
  - COBOL / CICS, Basic Mapping Support for terminal display to client-server architecture under OS/2.
  - Basic to Visual C++ (using a similar method)
- *Replacement* of the old user interface with the new

The static analysis technique implemented and validated in this research is very similar to part of the strategy outlined in my original proposal. The success of Merlo's group is encouraging and indicates that user interface components can be detected from legacy code using automated methods, strongly supporting my thesis.

Comparison to Proposed Research

Although the Macroscope findings will be very valuable to this research, there are still significant differences in the approach proposed here and that of Merlo's group. These have been elaborated above, but to summarize:

- The Macroscope approach uses purely static analysis to obtain and refine the model. I propose to use a combination of static and dynamic analysis in the recovery process.
- The Macroscope approach treats the user interface model as a single combined entity. I propose to extract several different interoperating models to restructure to a Seeheim architecture, in order to take advantage of a UIMS for forward engineering of the user interface.
- I propose to address detection of concurrency and higher level semantics such as context-sensitive availability of user interface objects which are not addressed in Macroscope.
- I propose to use a different representation method (Harel's State Charts) for the dialog control model rather than the process algebra-based representation used by Macroscope.

Revised Research Plan

This section replaces the "Research Plan" section of the original proposal. It outlines a plan for accomplishing this work, including focusing the research, identifying issues and questions, evaluating alternatives, and implementing a prototype solution to the user interface reengineering problem.

Research Focus

The main technical focus of this research will be to develop reverse engineering techniques and representation models for extracting components of the Seeheim user interface architecture from text-based systems in the Information Systems Domain. These systems are targeted because there

are many benefits in automating the reengineering process for these large, mostly text-based applications. The Seeheim model is chosen because it allows the legacy system to be restructured to take advantage of the power, flexibility, and abstraction of a UIMS.

The Mastermind UIMS system will be considered as a forward engineering environment, and the Mastermind Task Format (MTF) representation will be considered for the data model and also as the target of the transformation step. The MTF representation is still evolving, which presents the opportunity of potentially influencing the design of the representation to accommodate both forward and reverse engineering. Harel's State Charts will be used to represent the dialog control model, and the MTF object-oriented data modeling features will be incorporated into the State Chart representation.

Issues and Questions

There are many interesting issues and questions in this problem area. This work will address:

- Refining three sets of recognition rules for extracting dialog control components, application interface components, and presentation components from the legacy code
- Maximizing static analysis to detect user interface objects, actions, preconditions, postconditions, and semantics
- Utilizing dynamic analysis to complete the detection process, including instrumentation or interpretation of legacy code
- Developing a representation technique by incorporating a data and condition model into Harel's State Charts
- Giving the analyst control over decisions and the capability of adding human judgement to refine the model as the user interface model is built.

Validation and Evaluation Criteria

Validation will need to be performed in several dimensions for this work:

- *Productivity Gain* - A major emphasis of automated maintenance tools is the productivity gain. The time to reverse engineer a user interface with this automated tool will be compared against the time required to hand-reverse engineer the same application.
- *Quality of interface* - If the user interface model detected does not allow a high-quality user interface to be produced, then it is useless. For this validation, I will compare the automatically generated interface against hand-reverse-engineered systems described above in usability studies as a basis of comparison.
- *Scalability* - Since large size is a primary attribute of information systems, a large-scale reverse engineering task will be attempted to assess the scalability of the technique.
- *Analysis Technique* - The approach of combining static and dynamic detection will be compared against a purely static detection technique, as implemented in the Macroscope project.
- *Quality of Representation* - The models that result from the extraction process will be evaluated for human readability and understandability, and the power and flexibility of the representation technique will be evaluated against the AUIDL representation of the Macroscope project.

Plan

Deliverables for this work include:

- A set of recognition rules to perform detection on the presentation component model, the dialog control model, and the application interface model for the legacy application.
- Development and adaptation of an abstract representation for each of the three Seeheim user interface model components
- Investigation of transformation techniques to perform the application-control to dialog-control paradigm shift.
- A working research prototype for static and dynamic detection and representation of the complete model, investigating transformation into the Mastermind UIMS for forward engineering.
- Analysis of experiments with the prototype to assess performance, quality of detection, and functional equivalence to the original code.

Projected Schedule

Research Tasks:

- Develop and refine a dialog control model representation based on Harel's State Charts, adding a data model compatible with Mastermind MTF for presentation and application interface models. March 22
- Develop and refine recognition rules and express formally for each of the Seeheim model components:
  - Presentation Model April 5
  - Dialog Control Model April 26
  - Application Interface Model May 10
- Implement the static detection framework
  - Using Refine [REA94], implement the three rule bases May 31
  - Implement the extraction of the User Interface Slice from Abstract Syntax Trees built with Refine-based tools June 21
  - Implement a tool to apply the rules statically and generate the appropriate representation for each model July 12
- Implement dynamic detection framework
  - From information gathered in static analysis, build a tool to instrument the code at critical points of interest in the dynamic analysis phase Aug 9
  - Implement a dynamic rule recognizer Aug 30
- Develop a simple communication model between the user interface and the application to map values from the interface to the computational code Sept 20
- Investigate strategies for transforming the State Chart dialog control model into Mastermind MTF task descriptions. Oct 25
- Experiments with prototype toolset
  - Reengineer a small legacy system using both automated and manual methods and compare the results. Nov 8
  - Reengineer a large system using the prototype toolset to evaluate scalability Nov 29

Completion Criteria

This work will be considered to be complete when the deliverables mentioned above are completed and have been demonstrated to be valid, plus the development of the associated thesis documenting the research.

Contribution to the field

This work will contribute to the field of software maintenance in the following ways:

- It will increase the body of knowledge in program understanding, by introducing techniques for detecting user interface abstractions;
- It will improve the state of the art for software maintenance, and show significant productivity gains in the reverse engineering process;
- It will significantly improve application migration by providing a path for upgrading a user interface; and
- It will add to the body of experience with software transition techniques.

Conclusions

Program understanding and reverse engineering techniques for user interfaces goes far beyond simply replacing the old text-based user interface with the veneer of a new, flashy, graphical interface. Although the production of a new user interface is a primary goal, the information gleaned from the detection process can ease the entire process of reengineering. Developing an abstract model of a user interface can make the application more maintainable, portable, and usable.

While traditional reverse engineering techniques have concentrated on understanding program data structures and control flow, the user interface has been almost ignored. It is important to understand this facet of a legacy system in order to adequately reengineer it. Automating this process represents a significant savings in cost, time, and effort.

References

[DIX93] Dix, Alan; Finlay, Janet; Abowd, Gregory; and Beale, Russell. *Human-Computer Interaction*, Prentice Hall International (UK) Limited, 1993.

[GRE85] Green, Mark. "The University of Alberta User Interface Management System", *Proceedings of SIGGRAPH, 12th Annual Conference*, San Francisco, CA, July 1985.

[HAR89] Hartson, H. Rex, and Hix, Deborah. "Human-Computer Interface Development: Concepts and Systems for Its Management", ACM Computing Surveys, Vol 21., No. 1, March 1989.

[MER93] Merlo, E.; Girard, J.F.; Kontogiannis, K.; Panangaden, P.; and De Mori, R., "Reverse Engineering of User Interfaces" Proceedings of the Working Conference on Reverse Engineering, Baltimore, MD, May 21-23, 1993.

[MER95] Merlo, Ettore; Gagne, Pierre-Yves; Girard, Jean-Francois; Kontogiannis, Kostas; Hendren, Laurie; Panangaden, Prakash, and DeMori, Renato; "Reengineering User Interfaces" *IEEE Software*, Vol. 12 No. 1, January 1995.

[MOO94a] Moore, Melody. *User Interface Reverse Engineering,* a Research Proposal Draft, College of Computing, Georgia Institute of Technology, June 10, 1994.

[MOO94b] Moore, Melody. "A Technique for Reverse Engineering User Interfaces", *Proceedings of the 1994 Reverse Engineering Forum*, Victoria, B.C., Sept 1994.

[MOO96] Moore, Melody. *Representation Issues for Reengineering User Interfaces*, Research Report, College of Computing, Georgia Institute of Technology, Feb 19, 1996.

[REA94] Reasoning Systems, *The Refine Language Tools*, Marketing literature, 3260 Hillview Avenue, Palo Alto, CA, 94304, 1994.

[SUT78] Sutton, J., and Sprague, R. "A Survey of Business Applications", *Proceedings of the American Institute for Decisions Sciences 10th Annual Conference,* Part II, Atlanta, GA, 1978.