1995

Interaction and Interdependency of Software Engineering Methods and Visual Programming

Robert A. Touchton
University of North Florida

Suggested Citation

Touchton, Robert A., "Interaction and Interdependency of Software Engineering Methods and Visual Programming" (1995). *UNF Graduate Theses and Dissertations*. 217.
https://digitalcommons.unf.edu/etd/217

Interaction and Interdependency of Software Engineering Methods and Visual Programming Languages/Tools

by

Robert A. Touchton

A thesis submitted to the
Department of Computer and Information Sciences
in partial fulfillment of the requirements for the degree
of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

April, 1995

The thesis "Interaction and Interdependency of Software Engineering Methods and Visual Programming Languages/Tools" submitted by Robert A. Touchton in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the thesis committee: Date

Signature Deleted

5/2/95

Dr. Roger E. Eggen
Thesis Adviser and Committee Chairperson

Signature Deleted

5, 2, 95

Dr. Behrooz Seyed-Abbassi

Signature Deleted

5/2/95

Dr. Judith L. Solano

Accepted for the Department of Computer and Information Sciences:

Signature Deleted

5/2/95

Dr. Charles N. Winton
Chairperson of the Department

Accepted for the College of Computing Sciences and Engineering:

Signature Deleted

5/2/95

Dr. Charles N. Winton
Interim Dean of the College

Accepted for the University:

Signature Deleted

5-2-95

Dr. Charles Galloway
Dean of Graduate Studies

ACKNOWLEDGMENT

First and foremost, I thank my wife, Cheryle, my son, Chris, and my daughter, Kelley, not only for their support and tolerance, but also their enthusiasm and interest in this endeavor. I also appreciate the prayers and tender loving care that my church friends provided. They helped me maintain and balance my priorities.

My employer, PathTech Software Solutions, Inc., receives the credit for the inspiration and foundation for this effort. My partners and co-workers enthusiastically supported this undertaking with ideas, feedback and a solid grounding in the practice of software engineering.

I especially must thank Ken Wilson, Maureen Page, and Ray Schafer for the hours they spent with me as Peer Reviewers. They each added a real-world flavor to this effort along with a sense that the results can be of benefit to fellow software engineering practitioners.

Finally, I want to extend a special word of thanks to the CIS staff at UNF. I personally interacted with most of the Professors and found them to be not only knowledgeable and capable educators, but also truly interested in the

conveyance of knowledge and the improvement of the state of the practice of software engineering. The support staff was always helpful and professional. My thesis committee truly went the extra mile in their guidance and review of this effort.

CONTENTS

FIGURES

TABLES

ABSTRACT

Visual Programming Languages and Visual Programming Tools incorporate non-procedural coding mechanisms that may duplicate, or perhaps even conflict with, the analysis and design mechanisms promulgated by the mainstream Software Engineering methodologies. By better understanding such duplication and conflict, software engineers can take proactive measures to accommodate and, ideally, eliminate them. Better still, there may be opportunities for synergy that can be exploited if one is looking for them.

This research explored, documented and classified the interactions and interdependencies, both positive (synergies) and negative (conflicts), between two closely related and rapidly evolving Computer Science subdisciplines: software engineering and visual programming. A literature search was conducted to surface, evaluate, and build upon (where appropriate) recent and ongoing research in this area. A mechanism was created to capture observations of conflicts and synergies. This capture mechanism was applied to an experimentation test bed that was established to provide concrete examples of gaps, overlaps, conflicts, and synergies. In this regard, two relatively simple applications, one data-base oriented and one algorithm oriented, were designed and implemented using multiple software engineering methods and multiple visual tools/languages.

A software prototype, which bridges one of the gaps discovered during the research, was built to underscore the importance of eventually merging Computer Aided Software Engineering and visual development tools. The overall results as well as anticipated trends and developments in the area of software engineering and visual programming were summarized. The synergy/conflict observations, in conjunction with the literature search results, were used to develop strategies and guidelines for successfully using visual programming languages and tools in concert with sound software engineering methods.

Chapter 1

INTRODUCTION

Visual Programming Languages (VPLs), such as Visual Basic and Visual C++, and Visual Programming Tools (VPTs), such as ObjectVision and PowerBuilder, incorporate non-procedural coding mechanisms. Some of these mechanisms may duplicate, or perhaps even conflict with, the analysis and design mechanisms promulgated by the mainstream Software Engineering methodologies, such as Gane & Sarson or Coad/Yourdon [Pressman92]. By better understanding such duplication and conflict, software engineers can take proactive measures to accommodate and, ideally, eliminate them. Better still, there may be opportunities for synergy which can be exploited if one is looking for them.

VPLs and VPTs have been made possible by the maturation of Object-Oriented Programming constructs, leading to a strong correlation between the use of these new tools and OOP. Indeed, every one of the languages and tools studied rely strongly upon OOP for their internal design and operation (although they may differ in the degree to which they make OOP constructs available to application developers) [West92]. Therefore, this work was conducted with a backdrop of object-orientation. However, the focus is on identifying and resolving gaps, conflicts and synergies between the use of structured, formal Software Engineering methods and the use of VPLs and VPTs.

1.1 Statement of Problem

The importance and use of VPLs and VPTs is growing at a rapid pace both locally and nationally. A recent issue of Computer Magazine devoted over 50 pages to visual programming [IEEE95] In the past 36 months, the number of Jacksonville-based companies seriously using visual development software has climbed from perhaps one or two to dozens (based on a non-scientific review of Florida Times-Union classified ads which mention one or more of the recognized visual programming tools/languages). Similarly, a casual search of the internet for job postings which mention such languages and tools returns hundreds of hits (just looking at the IEEE Careers, Career Mosaic and the "Monster" Board on the world wide web). At the same time, more and more companies are adopting formal software engineering methodologies, usually in the form of a commercially offered CASE (Computer Aided Software Engineering) tool. Personal experience gained in the author's work environment has increased his awareness of inconsistencies between these two Computer Science subdisciplines. He also became convinced that positive steps can and should be taken to ensure that the benefits from both of these technological advancements are realized. This conviction forms the basis of this research.

The need for substantial advances was foreshadowed in Lowry's 1992 article in AI Magazine where he suggested that current CASE tools were shallow, that the latest programming environments were good for prototyping but lacked the ability to produce efficient, production-quality code and that perhaps the use of artificial intelligence could close the gap [Lowry92]. More recently, the gap between CASE tools and implementation tools have been editorialized in software engineering trade journals. For example, in one issue of Software Development magazine, Larry Constantine emphasizes the importance of one day being able to program by drawing models of the target application[Constantine94] and Larry O'Brien points how event-driven architectures, visual programming aids and the like have seriously challenged the traditional CASE tools[O'Brien94].

Inconsistencies may manifest themselves as conflicts, gaps, or overlaps in screen layout, process diagrams, Entity-Relationship Diagrams, or Data Dictionaries. Synergies take shape as opportunities for direct program generation and rapid prototyping, as well as improved communication with end users; the day may come where end-users can use a VPT to build their own prototype as a starting point for implementation by a central IS group.

An example of a conflict would be writing a traditional Program Flow Chart only to find out that the VPT must approach the flow of control in a completely different way. Consider the fact that ObjectVision relies on two event-driven program flow mechanisms, neither of which have a direct mapping from a traditional flow chart: "when-changed" methods attached to data elements and "logic trees" for responding to user- and application-generated messages. The flow chart, while useful for communicating desired program behavior, will provide little or no insight about how to implement that behavior. A similar case can be built for pseudocode. To turn the example into one of synergy, imagine that the designer had been able to access a tree-like representation to devise and communicate the program flow of control. This would provide insight into its implementation in addition to its desired behavior. Perhaps the software engineering method could be modified to actually embrace the visual event trees of ObjectVision as its program flow representation.

1.2 Research Plan

During the formative stages of this research, the author proposed a definitive series of steps aimed at ensuring that the effort would be of graduate-level quality and content and that the objectives of the effort would be achieved. Upon consultation with the Thesis Committee, a final research plan was established, as reflected in the following steps:

1. Devise a problem classification scheme and mechanism for capturing and documenting conflicts (e.g., gaps or overlaps)and synergies (smooth transitions and cooperation) between software engineering methods and implementation languages and tools

2. Review and evaluate modern software engineering methodologies

3. Review and evaluate visual programming languages and tools

4. Create a controlled experimentation test bed to provide concrete, working examples of interactions and interdependencies in the form of gaps, overlaps, conflicts, and synergies

5. Develop a prototypical bridge for generating Graphic User Interface Screens in the "native tongue" of a visual programming language based on screens designed in a CASE tool
6. Evaluate and Document Results, including a summary of major trends and guidelines for avoiding problems and ensuring synergies

The objective of this research was to explore, document and classify the interactions and interdependencies, both positive (synergies) and negative (conflicts), between two closely related and rapidly evolving Computer Science subdisciplines: software engineering and visual programming.

1.3 Literature Search

One element of the research effort was conduct of a literature search to surface, evaluate, and build upon (where appropriate) recent and ongoing research in this area. Much of the fruit of that search is embodied in this thesis. Most of the remainder of the information found was of general or bibliographic value, but not suitable for direct reference.

The literature search included Journals, such as the Journal of Visual Languages and Computing and IEEE Transactions on Software Engineering, Transactions/Proceedings, such as ACM Transactions on Programming Languages and Systems, Texts, such as Object Oriented Design with Applications by Booch [Booch91], and Code Complete, by McConnell [McConnell93]. The effort also included a search of the internet, using archie.

1.4 Summary of Results

Chapter 2 describes the problem classification scheme and observation collection mechanism called for in the first Step. Problem groupings, as well as areas of cooperation, arising between the software engineering methods and the visual programming languages and tools are identified. A data base tool to facilitate the capturing of conflict and synergy observations is also described.

The results of the second step of the Research Plan are presented in Chapter 3. The treatment briefly discusses those traditional and object-oriented software engineering methods used in this study, along with a summary of their notation and use. A section on the CASE tool used for portions of this effort is also provided.

The results of step 3 are included in Chapter 4, which provides a synopsis of the visual programming languages and tools evaluated along with an overview of visual programming. The treatment briefly compares and contrasts the various tools and languages and, for ones not used in the test bed, provides specific examples of areas which make these software packages unique. Since the maturity of the desired guidance required immersion in the selected language/tools, attention was paid to the mechanics and details of their operation. Generalizations could then be drawn from the specific experiences in their use.

During step 4, two test applications were defined: one data base-oriented and one algorithmic in nature. Then, two software engineering methods were selected, one used for each application (see Table 1). Finally, one visual programming language and two visual programming tools were selected and each application was implemented using the VPL and one VPT (see Table 1). As the applications were designed and implemented, the capture mechanism from step 1 was applied. Table 1 summarizes the test bed resulting from step 4 of the Research Plan, while Chapter 5 and Appendices A through F present it in detail.

The software bridge between the System Architect CASE tool and Visual Basic, resulting from step 5, are presented in

Chapter 6 and Appendix G. The bridge points to the importance of eventually merging Computer Aided Software Engineering and visual development tools.

The results of step 6 are presented in Chapter 7 and Appendices H and I. Because of the impact on this research of the rapidly changing software development environment, a treatment on current trends and developments was compiled. The synergy/conflict observations, in conjunction with the literature search results, were used to develop strategies and guidelines for successfully using visual programming languages and tools in concert with sound software engineering methods. The guidelines were segmented based on type of problem being addressed. Finally, the overall effort and its contribution to future efforts are summarized.

Application Type	Application Name	Software Engineering Methodology	Implementation Language	Implementation Tool
Data Base - Oriented	Customer Problem Tracking System	Gane & Sarson DFDs plus ERDs	Visual Basic	ObjectVision
Algorithm - Oriented	Tic-Tac-Toe	Coad/Yourdon	Visual Basic	Smart Elements

Table 1: Experimentation Test Bed Matrix

Chapter 2

Conflict/Synergy Capture Mechanism

During the planning stages of the research, it was recognized that there needed to be a formal mechanism in place to assist the author in capturing and distilling the examples of conflict and synergy that were to be sought ("self observations"). Therefore, the premise that conflict and synergy should be detectable was carried to more specific categories, as discussed in Section 2.1. These classifications would then be used to create a data base which would serve both as a repository of experiment observations and as a prompting device to elicit a consistent slate of information about each observation (Section 2.2). As the research moved into the experimentation phase, it was further decided that the observation data base would be altered slightly to enable it to be used to elicit information from working practitioners using similar combinations of CASE tools and VPTs ("peer observations").

2.1 Conflict/Synergy Classifications

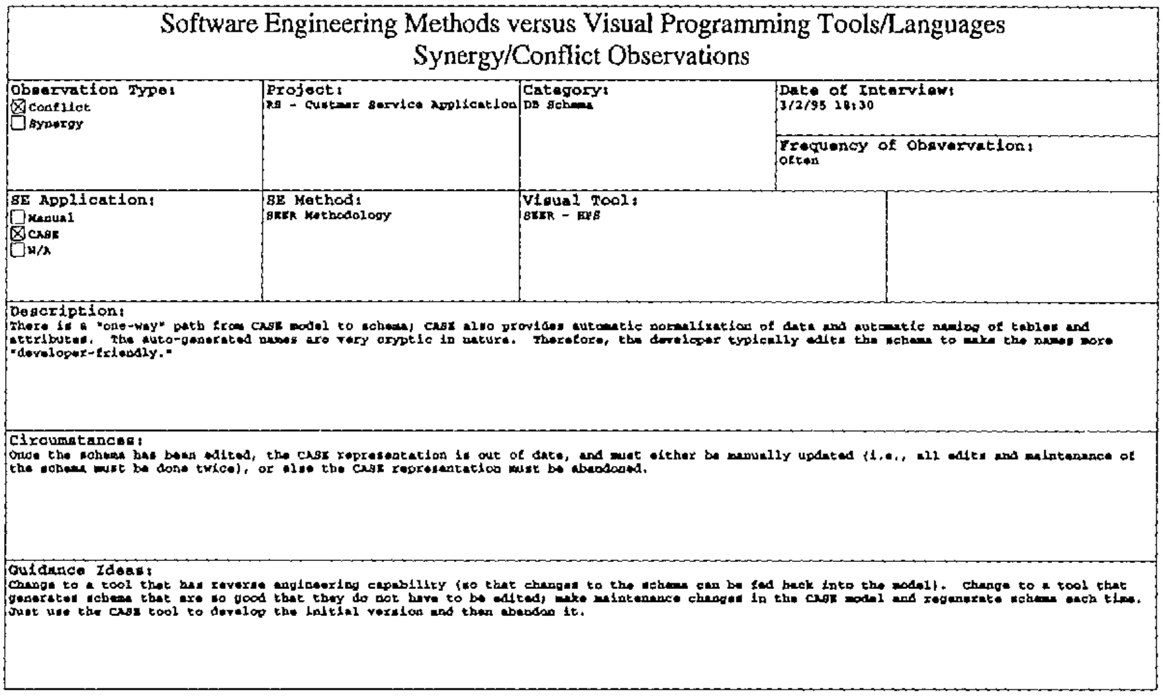
The issues that might be encountered during visual development were classified into two major categories, Structural and Behavioral, each with three subcategories. As shown in Table 2, types of conflicts and synergies were identified for each subcategory (where applicable) and specific examples given.

Category	Conflict	Synergy	Examples
STRUCTURAL:			
User Interface Layout	Duplication of Effort	Automation of Effort	(-) Sketch in CASE (or on paper), then re-sketch in the tool or language (+) Sketch in CASE with generation of screens in the tool or language
DB Schema Design	-	Automation of Effort	(+) Design Schema in CASE with generation DB structures in the tool or language
	Tool/Language Does Not Support the Schema	-	(-) Relational Design, but tool or language only supports "flat"
Object Representation	Tool/Language Does Not (fully) Support	Full Object-Oriented Environment	(+) Developer-definable objects/attributes with multiple inheritance, such as C++ (-) Limited ability to create/use objects, such as Visual Basic
BEHAVIORAL:			
Service/Utility Modules	Designing Modules that are already "Built-In" to the Tool/Language	Utilizing Them	(+/-) File Browser; Print Setup
Function/Routine Design	Tool/Language Does Not Support the Concept	Tool/Language Strongly Supports the Concept	(+) Time Object in VB (-) Dealing with time in OV
Event-Based Design	SE Methods are limited	Strong Support from Modern Tools and Languages	(+/-) See Pressman's Watch example on [Pressman92, pp. 495-497]

Table 2: Visual Development Conflicts/Synergy Matrix

2.2 Conflict/Synergy Observation Data Forms

The content of the classification scheme was used to derive a series of data elements and types of textual information that ought to be collected for each observation of a conflict or synergy. These elements were used in turn to create a data base and corresponding GUI. Figure 1 shows a typical data form as it looks in the GUI. ObjectVision, which was used to create the application, has the ability to print out the data base as a series of sheets that look like the data entry form. The possible or likely values of the data elements were used to populate pull-down menus and to create the check boxes seen in the figure.



Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type:
☒ Conflict
☐ Synergy

Project:
RS - Custmer Service Application

Category:
DB Schema

Date of Interview:
3/2/95 18:30

Frequency of Obsvervation:
Often

SE Application:
☐ Manual
☒ CASE
☐ N/A

SE Method:
SEER Methodology

Visual Tool:
SEER - HPS

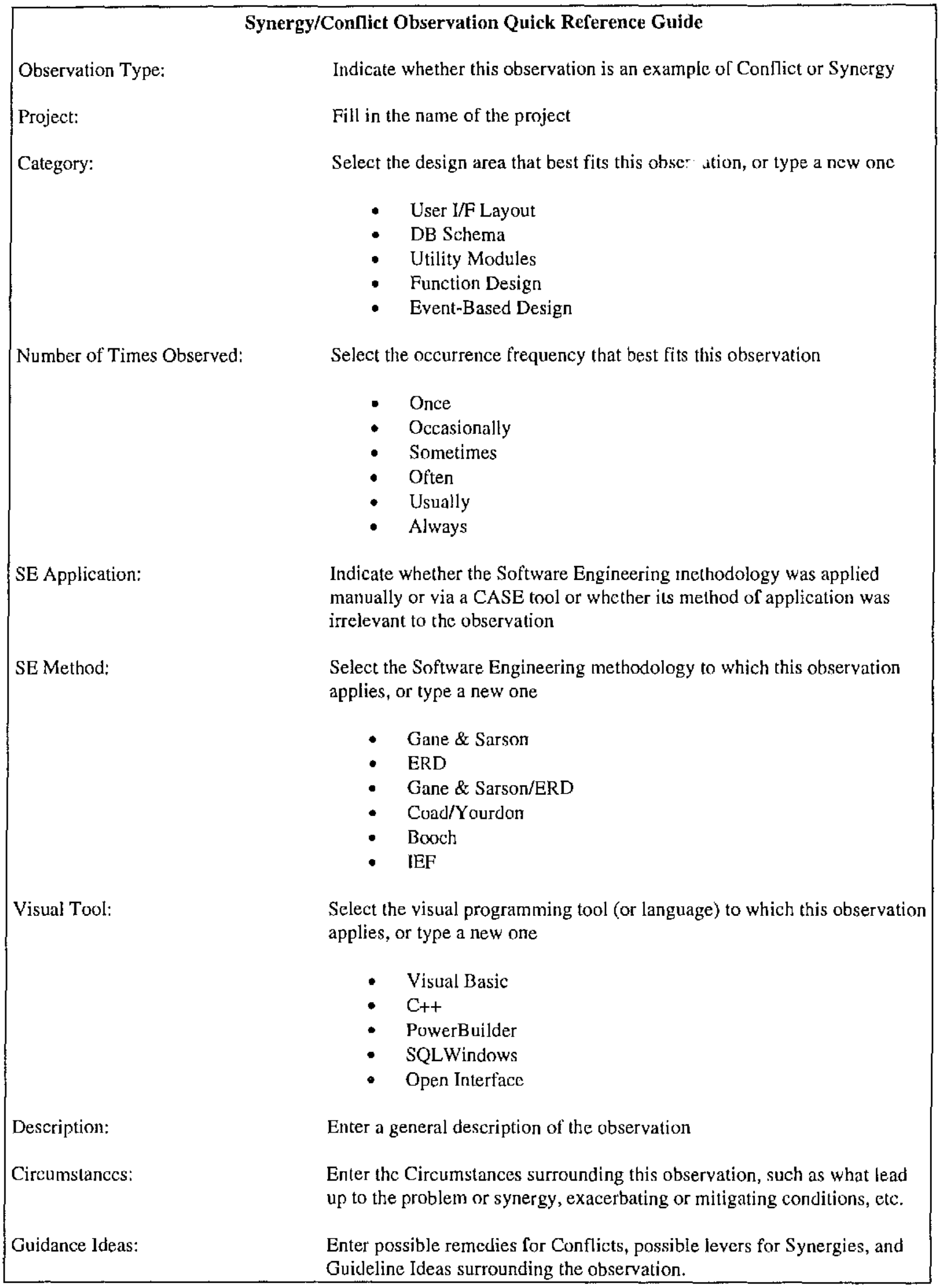
Description:
There is a "one-way" path from CASE model to schema; CASE also provides automatic normalization of data and automatic naming of tables and attributes. The auto-generated names are very cryptic in nature. Therefore, the developer typically edits the schema to make the names more "developer-friendly."

Circumstances:
Once the schema has been edited, the CASE representation is out of date, and must either be manually updated (i.e., all edits and maintenance of the schema must be done twice), or else the CASE representation must be abandoned.

Guidance Ideas:
Change to a tool that has reverse engineering capability (so that changes to the schema can be fed back into the model). Change to a tool that generates schema that are so good that they do not have to be edited; make maintenance changes in the CASE model and regenerate schema each time. Just use the CASE tool to develop the initial version and then abandon it.

Figure 1: Typical Observation Data Form

During the completion of the experimentation test bed (see next Chapter), the "self observation" forms were filled out soon after each conflict or synergy was encountered. The "peer observation" version of the forms were used in an interview setting. The author described the premise and nature of the study, presented the preliminary results, and then encouraged the participant to fill out a data form for each memorable example of a conflict or synergy within the scope of the research. A "Quick Reference Guide," shown in Figure 2, was developed to help peers when filling out the forms.



Synergy/Conflict Observation Quick Reference Guide

Observation Type: Indicate whether this observation is an example of Conflict or Synergy

Project: Fill in the name of the project

Category: Select the design area that best fits this obser ation, or type a new one

- User I/F Layout
- DB Schema
- Utility Modules
- Function Design
- Event-Based Design

Number of Times Observed: Select the occurrence frequency that best fits this observation

- Once
- Occasionally
- Sometimes
- Often
- Usually
- Always

SE Application: Indicate whether the Software Engineering methodology was applied manually or via a CASE tool or whether its method of application was irrelevant to the observation

SE Method: Select the Software Engineering methodology to which this observation applies, or type a new one

- Gane & Sarson
- ERD
- Gane & Sarson/ERD
- Coad/Yourdon
- Booch
- IEF

Visual Tool: Select the visual programming tool (or language) to which this observation applies, or type a new one

- Visual Basic
- C++
- PowerBuilder
- SQLWindows
- Open Interface

Description: Enter a general description of the observation

Circumstances: Enter the Circumstances surrounding this observation, such as what lead up to the problem or synergy, exacerbating or mitigating conditions, etc.

Guidance Ideas: Enter possible remedies for Conflicts, possible levers for Synergies, and Guideline Ideas surrounding the observation.

Figure 2: Peer Observation Handout

Chapter 3

Software Engineering Methodologies and CASE Tools

Software Engineering can be defined as "the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines," and comprises three key elements: methods, tools, and procedures [Pressman92, pp. 23-24]. There is a wide variety of Software Engineering techniques and methodologies in current use by Computer Scientists in both academic and commercial settings. These techniques and methodologies address a wide range of issues, including project planning, management and estimation, software quality and testing and detailed software design and implementation. However, the focus of this investigation is the areas of requirements analysis and high-level design. Some methods are very formal and structured, while others are more heuristic in nature, providing good practice guidelines.

The sections that follow highlight the software engineering methods and tools used in conducting the research for this thesis. Section 3.1 provides a general synopsis of the various methods and tools evaluated. Section 3.2 addresses the more traditional techniques used to support the data base applications. Section 3.3 looks at the object-oriented/event-based techniques used for the algorithmic applications. Section 3.4 explores the CASE tool used for portions of the work.

3.1 Synopsis of Software Engineering Methodologies and CASE Tools Evaluated

Each method has associated with it a set of tools, whether applied manually, automatically, or both, which manifests a special language or graphical notation. The more widely used methods have been integrated with one another and incorporated into Computer Aided Software Engineering (CASE) tools. Two of the most widely used methods, Data Flow Diagrams and Entity-Relationship Diagrams, are described in section 3.2. Several other mainstream methods were evaluated for use in the data base-oriented application but were eliminated based on either unnecessary complexity for the target problem or focus upon areas irrelevant to the target problem. For example, a State Transition Diagram (STD) can be developed to identify the possible states of each data entity and the allowed transitional events that can cause the state to change. However, an STD for the Customer Support Tracking System would have been trivial and of little value to the research. Likewise, a Ward and Mellor real-time analysis can be conducted to identify time-based processes and interfaces to physical devices. However, the intended system exhibited no complex real-time behaviors, rendering a Ward and Mellor analysis useless.

Several object-oriented software engineering methods were also evaluated. Coad/Yourdon, the one selected for the Tic Tac Toe game, is described in section 3.3. Two other methods, Rumbaugh and Booch, were investigated. Each of the three has its own style and approach to object-oriented analysis and design, with no one method appearing stronger than the others. Therefore, the selection of Coad/Yourdon was primarily based on the author's prior exposure to it.

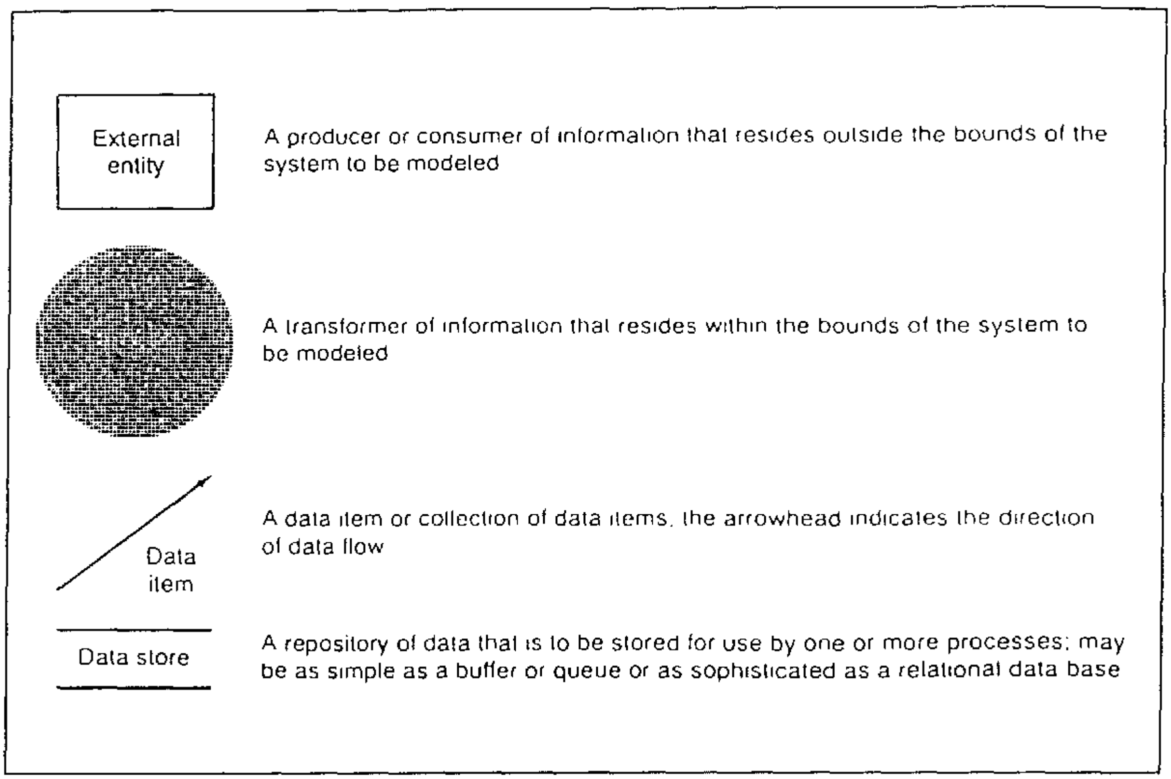
Commercial-grade CASE tools are now widely available, with a wide range in both capabilities and price. High-end tools, such as IEF by Texas Instruments and HPS by SEER, are extremely robust and powerful within the framework of their intent (e.g., legacy or mainframe data bases), and typically include a high degree of support for team programming and automation of production tasks. Such tools also bring with them a high per-developer price tag, typically above $10,000. Unfortunately, the high-end tools having any significant market share have their roots in mainstream, conventional programming environments with little to no support for object-oriented programming environments, event-based programming, Graphic User Interfaces or 4GLs (exceptions to this statement are discussed in Section 7.2)

Smaller, more specialized or single-purpose tools are also becoming more widely used, such as VisSim by Visual Solutions Inc. VisSim, a tool that sells for under $200, allows engineers to create on-screen diagrams that model and simulate physical processes. In between is a class of tools whose members are priced in the $1000 - $2000 per-developer range and provide an assortment of the more popular Software Engineering methods. Tools in this category include ERWin, EasyCASE and System Architect. The capabilities of System Architect will be discussed further in Section 3.4 since it was used during the course of this research. Another category of CASE tools encompasses those provided by data base vendors. For example, ORACLE now provides an excellent suite of software engineering tools as add-on products to their popular relational data base product.

3.2 Gane & Sarson Data Flow Diagrams and Entity Relationship Diagrams

The primary method used for analyzing and designing the Data Base Management System (DBMS) aspects of the test bed was the creation of Data Flow Diagrams (DFDs). The specific DFD notation style of Gane & Sarson was adopted. The use of DFDs for analysis and design of information systems is well documented in a number of software engineering texts [Pressman92, Chapters 7 and 11]. For purposes of this thesis, it is sufficient to summarize that DFDs provide a structured methodology for representing external sources and sinks of data, the processes that manipulate the data, stores of data, and the specific data that must flow to and from each process. Figure 3 shows the graphical notation for each of these items. The methodology relies heavily on levels of abstraction, such that a process modeled in general at one level can be examined in greater detail at another level. The zeroeth (most abstract) level is referred to as the "context" diagram, showing only one process: the software application under study. The level 1 DFD breaks the application down into its major modules, level 2 divides those modules into smaller components, and so it goes until the modeler is satisfied that the processes shown on the diagram are sufficiently atomic to be implemented. Although highly subjective, the dividing line between analysis and design is often set between level 2 and level 3 DFDs.

The second software engineering method used in the test bed was that of data modeling, using a powerful modeling tool known as the Entity-Relationship Diagram (ERD) [Pressman92,



A producer or consumer of information that resides outside the bounds of the system to be modeled

A transformer of information that resides within the bounds of the system to be modeled

A data item or collection of data items, the arrowhead indicates the direction of data flow

A repository of data that is to be stored for use by one or more processes; may be as simple as a buffer or queue or as sophisticated as a relational data base

Figure 3 Gane & Sarson DFD Notation [Pressman92, page 210]

section 8.3]. ERDs pick up where DFDs leave off in terms of detailing the form, content and structure of the relevant data elements. Specifically, ERDs identify each data entity, its attributes, and how it relates to other entities in terms of cardinality and function. ERDs are especially useful when the design calls for a relational data base: the entities become tables, the attributes become column names, and the relationships become cues to the required referential keys. Figure 4 shows the ERD graphical notation used for this effort.

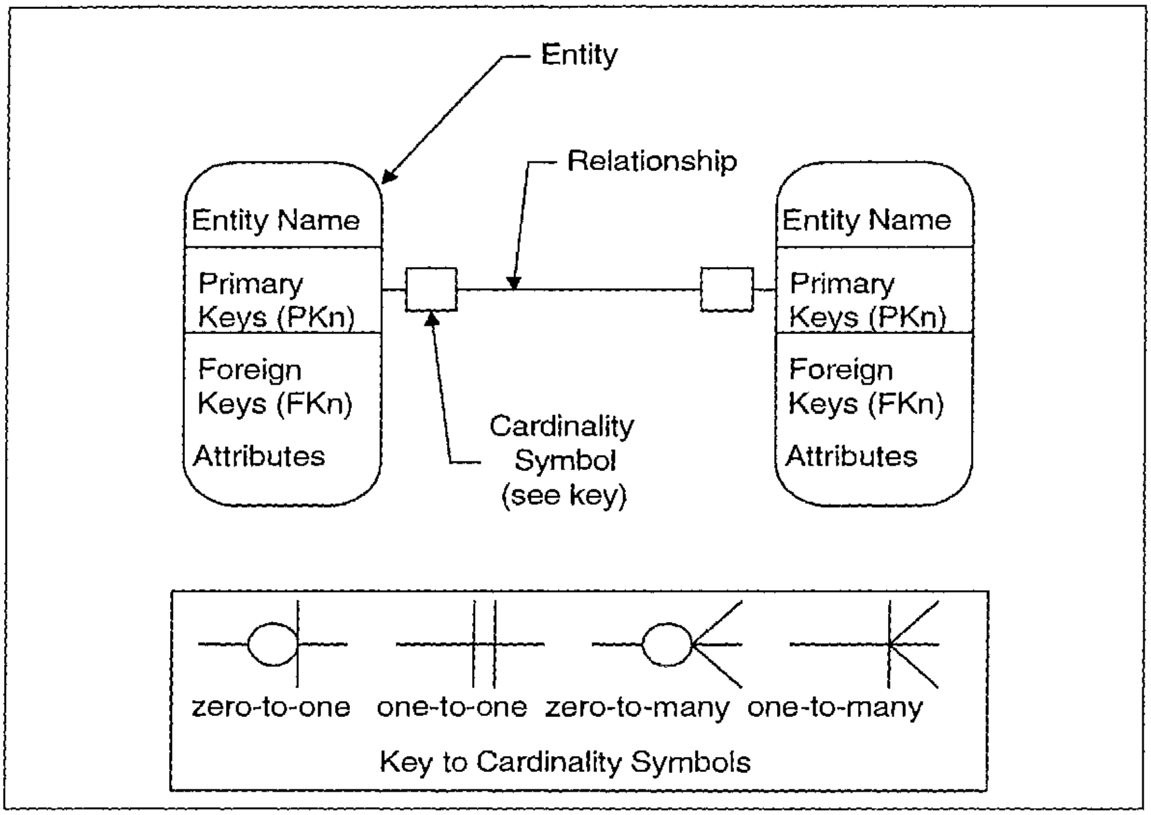


Figure 4: ERD Notation

The transition from DFD to ERD takes place at the DFD Data Store. In other words, the next level of abstraction for a DFD Data Store is not another (higher level) DFD, but rather, is an ERD. In this fashion, the DFDs govern the DBMS (with emphasis on "management system") while the ERDs govern the Data Definitions (or structures) to be manifested in the data base.

3.3 Coad/Yourdon

Object-Oriented Analysis and Design (OOA/D) was the primary method used for the algorithmic portion of the test bed (i.e., the Tic Tac Toe game). The specific method and notation style of Coad/Yourdon was adopted [Coad/Yourdon90 and Coad/Yourdon91]. OOA/D was considered well-suited as one component of the test bed because the visual tools targeted for implementation were known to be object-oriented and event-based in nature. For purposes of this thesis, it is sufficient to summarize that OOA/D provides a structured methodology for representing classes and objects, their interconnecting structure, their attributes, the services they provide and the message connections that communicate the need for services. Figure 5 summarizes the graphical and semantic notation used in the Coad/Yourdon approach to OOA/D. Note that the abstraction tool referred to as "subject" was not required for this effort.

The Service Chart notation was also extrapolated to model the overall behavior of the application. Thus, the notation was used to provide a flow chart style representation of an event-based design.

The procedural aspects of the OOA/D effort followed Coad/Yourdon's OOA Strategy Summary and OOD Strategy Summary [Coad/Yourdon91, pp. 164-181 (Appendix B and

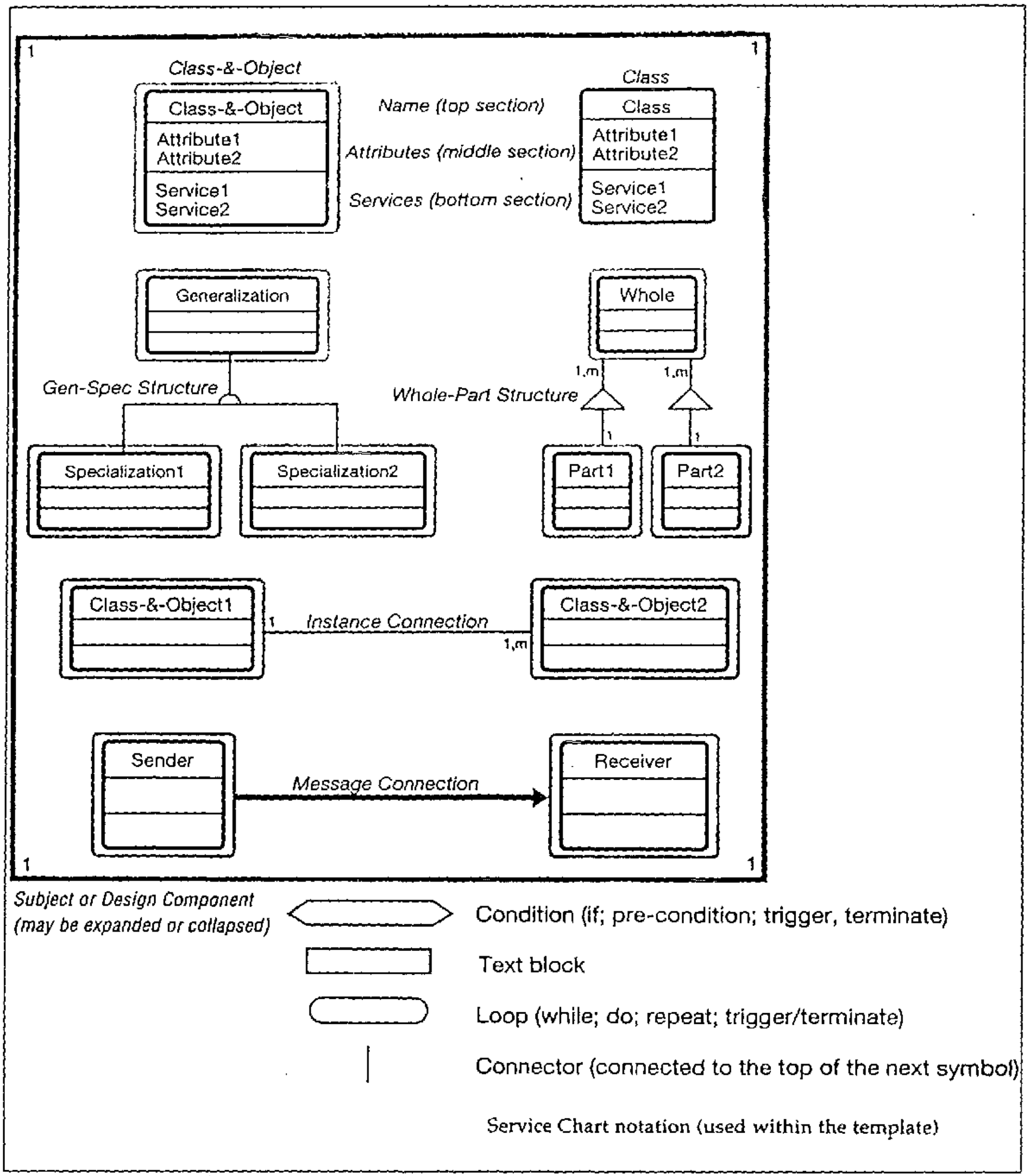


Figure 5: Coad/Yourdon OOA/D Notation
[Coad/Yourdon91, page 162]

Appendix C)]. Although some of the strategy elements were not relevant to the simple test bed application, the sequence and content of the remainder were found to be quite useful.

3.4 System Architect CASE (Computer Aided Software Engineering) Tool

The System Architect CASE tool (version 3.0) was used to implement the software engineering methodologies discussed in Section 3.2 [SysArch94A]. (The OOD/A methods were applied manually.) Like most mid-range CASE products,System Architect is a desktop workstation-based suite of tools aimed at helping software developers provide higher quality, more efficient work products. It maintains an integrated, team-oriented repository (data base) of results, referred to as an "encyclopedia," which contains project-wide Data Definitions, Diagrams, etc. System Architect provides diagramming/modeling support and rules checking for a wide variety of popular software engineering methods, including all of the ones discussed in this thesis (the Object-Oriented methods are obtained via an optional upgrade module). It also provides, as upgrade options, a project report generator, a data base schema generator, a PowerBuilder bridge, and a reverse engineering tool. Another optional module which was used in this effort is the Screen Painter, which is a screen design and layout tool with a Windows Dialog generator (see Chapter 6).

The mechanics of using System Architect are straight forward. First, one must select the software engineering method to be used. The tool then provides an on-screen template of graphic icons relevant to that method which can be "dragged" and "dropped" onto the diagram under construction. Connection icons are further enabled to allow the developer to click on the icons to which they are connected with knowledge of directional flow (i.e., "from" the first icon clicked "to" the second one). Each icon can then be further refined by opening a series of "behind the scenes" dialog boxes (using a right-click to pop up a menu of options). These refinement dialogs handle duties from the routine, such as names and labels, to the advanced, such as cardinality and composite data definition statements.

System Architect also supports levels of abstraction by allowing each icon to link to a Child Diagram, which represents a more detailed breakdown of its parent. One of the rule-checking features of the tool is to verify that the inputs and outputs of a Child Diagram are consistent with those of the Parent (so-called "balancing").

Chapter 4

Visual Programming Languages and Tools

"Visual Programming" has been defined as "the use of meaningful graphic representations in the process of programming" [Shu88, page 9]. Shu further defines a visual programming language as one "which uses some visual representations (in addition to or in place of words and numbers) to accomplish what would otherwise have to be written in a traditional one-dimensional programming language" [Shu88, page 138]. A visual programming tool can be thought of as a higher-level development environment, incorporating a 4GL or scripting language and perhaps project management aids, interpretive testing (i.e., without compilation) or team development aids.

Visual programming (at least in a commercial setting) is tightly connected to object-oriented programming in that all of the tools and languages evaluated were themselves object-oriented and most allowed developers to enjoy the benefits of object-oriented programming to some degree. Further, visual programming enables object-oriented practitioners "to build applications from simple, reusable parts...by providing palettes of compatible parts in easily accessible formats ready for use by developers" [Jicha94]. Classification as a visual programming language or tool is not absolute, but is by degree. The Gartner Group suggests a continuum ranging from "Visual GUI with Text Scripts" (e.g., Visual Basic) to "Visual with Minor Text Required" (e.g., Smart Elements) to "Visual with Text Optional" (e.g., ObjectVision) to "Visual Only" (e.g., Layout) [West92]. Shu further decomposes the "Visual Only" category into "Diagrammatic systems" which use as their programming paradigm "flow charts and diagrams that are already in use on paper" (e.g., Layout), "Iconic systems" wherein "graphical symbols are deliberately designed to play the central role in programming" (e.g., G2) and "Form systems" which employ graphical software representations of physical tables and forms which "are designed as an integral part of a language" (e.g., Visual Basic) [Shu88, pp. 12-16].

4.1 Synopsis of Visual Languages and Tools Evaluated

Because of the variety and depth of visual programming languages and tools available, one task of this research was to evaluate a reasonable sampling of them and select the ones to be used in the experimentation test bed. This section provides a brief synopsis of each language/tool evaluated. The sections that follow elaborate on the ones actually used. It is important to note that the languages and tools evaluated as part of this research are representative of the class of visual programming languages and tools, thus allowing one to reasonably generalize the results of this study.

Visual Basic is a Microsoft product which provides a WYSIWYG layout tool for quickly constructing functional Windows front ends. It is somewhat object-oriented in that each visual element has self-contained attributes and behaviors; events which occur at run-time can trigger BASIC functions and procedures. However, programmers cannot define their own classes/objects or inheritance schemes. The visual nature of Visual Basic is laying out of the user interface and using the interface objects to organize and contain much of the behavior of the application. Visual Basic is an economical development language suitable for small to medium applications and is supplemented by a large catalog of third-party add-on tools and utilities. Section 4.2 elaborates on Visual Basic and how it was used to support this effort.

ObjectVision is a Borland product which offers a form-oriented WYSIWYG user interface layout tool, a tree-like visual language for processing logic, a spreadsheet-like macro language for manipulating data and a user-friendly "point and click" tool for linking data base files/tables with ObjectVision objects. If an application's requirements align with such features, ObjectVision can be an extremely powerful tool; conversely, attempting to build an application for which a form-style user interface is not appropriate, or one which requires procedural algorithms, can prove to be frustrating and non-productive. Like Visual Basic, it is somewhat object-oriented in that each visual element has self-contained attributes and behaviors, and events which occur at run-time can trigger ObjectVision functions and procedures; however, programmers cannot define their own classes or objects. ObjectVision offered an extremely economical price point, but is now being phased out by Borland. Therefore, it should not be used for any production-grade application where upgrades or support would be required. Section 4.3 elaborates on ObjectVision and how it was used to support this effort.

Smart Elements, by Neuron Data, originated as an expert system development tool named Nexpert Object (circa 1987). Later, they added a GUI development tool named Open Interface (circa 1991). Most recently, they bundled and integrated these two tools to form Smart Elements. Smart Elements is completely object-oriented in both its own implementation and its use by developers. It is also one of the most portable development environments available today in that even the most graphical application developed on one platform can be immediately recompiled on another (e.g., develop on DOS/MSWindows and deploy on Unix/Motif without touching the source). Smart Elements is visual in several ways. First, its GUI editor provides much the same functionality as discussed for Visual Basic, with the added benefit of extendibility and full object-orientation (e.g., one can create new widget classes and inherit from them). Next, much of the behavior of the application is implemented by filling out "point and click" and "fill in the blank" dialog boxes. Last, class/object/attribute/method hierarchies (as well as rule-bases) can be visualized as graphical tree structures. Smart Elements provides its own scripting language for high-level users (analogous to BASIC for Visual Basic), while allowing full-fledged software engineers to implement module details in C (analogous to Visual C++). This tool is moderately-priced. Section 4.4 elaborates on Smart Elements and how it was used to support this effort.

Visual C++ is a Microsoft product which provides a very robust class library for handling such diverse tasks as OLE support, graphical drawing, printing services. It also has the "Class Wizard" which handles the details of creating subclasses and instances, such as inserting the correct properties and behaviors and adding comments like "//ADD APPLICATION SPECIFIC CODE HERE." Visual C++ is by far the most powerful and robust of the tools evaluated. It is unfortunately the least visual; for example, it has a limited WYSIWYG layout tool when compared to Visual Basic.

PowerBuilder is a tool from PowerSoft for application developers who are creating the "client" end of client/server applications. It is geared towards MSWindows applications networked to a SQL data base server. PowerBuilder provides a WYSIWYG user interface layout tool, a non-standard scripting language, automatic or manual generation of SQL statements to put/get/manipulate server data and hooks into the C language. Although not fully open to programmers, its flexibility and power go well beyond that of Visual Basic and ObjectVision but at about 30 times the price. SQLWindows by Gupta serves much the same audience and provides much the same functionality as PowerBuilder. However, SQLWindows is considerably more open and extensible, probably due to the fact that is more object-oriented. SQLWindows has a slightly more attractive price point than PowerBuilder. Both tools offer team development add-ons and are well-suited for large-scale, production-grade development projects.

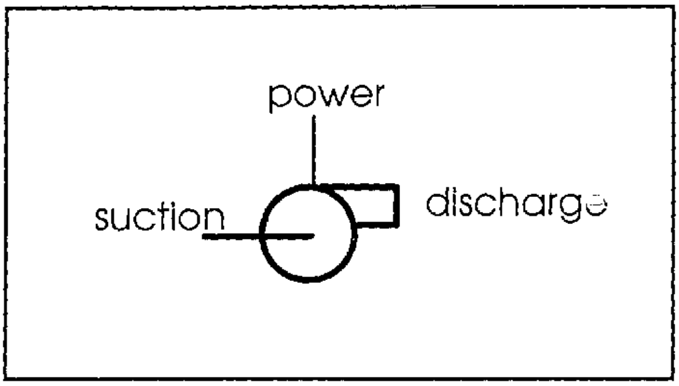


Figure 6: Iconic Object

G2 by Gensym, Inc. is a real-time expert system development shell which has a strong visual-programming component, especially for modeling and simulating physical systems such as process plants and factories. It supports Model-Based Reasoning in conjunction with visually laying out and connecting the components of the physical system. Each class of physical object is manifested in a G2 class object, including its visual icon, its connection ports, its possible states and its behaviors (usually in the form of an equation). Thus, the class, PUMPS, might have the icon depicted in Figure 6 with a suction port, a discharge port and a power port; states of pumping, available and out-of-service, and a behavior of "If the POWER SUPPLY object connected to the power port is energized, AND If the FLOW LINE object connected to the suction port is open, AND If the FLOW LINE object connected to the discharge port is open, Then the state of this PUMP is pumping."

Such class objects can then be instantiated, dragged, dropped, named and interconnected on a workspace using the mouse. G2 is very object-oriented and open (considering that it is a tool and not a language) but is very expensive ($50,000 to $100,000, depending on hardware platform and options selected).

The most visual tool examined was VisualAge by IBM. Because of the high price tag (and the fact that the author's place of work has not yet purchased a copy), the examination was at IBM at the hand of an IBM demonstrator. Nonetheless, it was obvious that VisualAge is a fully-functional visual programming tool. It is completely object-oriented in both its implementation and its use. It is built in Smalltalk and is extensible using Smalltalk. VisualAge was originally introduced in OS/2, but is now available in Windows as well. The tool has a GUI layout scheme much like the other tools, but carries forward the visual programming paradigm to include flow of information and control. The behavior of the application is programmed by dragging, dropping and connecting functional components and then adding any necessary conditionals or parameters. Were it not for the economic barrier involved, this author would have included VisualAge as one of the VPTs used in the experimentation test bed.

Layout from Objects, Inc., is the most unique of the VPTs considered. It uses the flow-charting concept to visually construct a working (and compilable) application. "Flow charts" are made up of user-connected "black boxes," each designed to provide a specific function (such as opening a window, displaying information, accessing a file, etc.). The basic program comes with over 200 pre-defined black boxes and allows users to assemble black box abstractions (called "procedures"), plus Layout allows a professional programmer to build additional ones in C/C++, Turbo Pascal and QuickBASIC. It claims that its compilation process uses an expert system to generate optimized final source code (in C/C++, Pascal or BASIC), rather than blindly append code fragments based solely on how the developer laid out the flow chart of black boxes. Objects Inc. purports that "Layout is probably the ultimate CASE tool...a full life-cycle CASE tool, able to assist you in diagramming and designing your program, prototyping it, fleshing it out, testing it, and then, when you're done, create the finished program for you." [Layout92] Layout is relatively inexpensive and supports DOS and MSWindows. Layout was originally targeted for use in the test bed; however, it turned out to be unsuitable for building an algorithmic application under Windows. However, Section 4.5 shows an example of how recursion is implemented in a visual programming tool.

There are other excellent visual programming languages and tools in commercial use, such as Forte' (Forte' Software), ObjectView (KnowledgeWare), VisualWorks (ParcPlace) and several offerings from Computer Associates [Hanna94]. There are numerous experimental and developmental ones as well, such as PFG (U. of Maryland), PT (U. Of Kansas), and HI-VISUAL (Hiroshima University) [Chang90] [Ichikawa90]. The omission of any tool or language from this treatment is not an indication of its value, but only a necessary limitation of the scope of this research effort. The sections that follow provide additional details for those tools actually used to support this thesis.

4.2 Visual Basic

Microsoft Visual Basic (version. 3.0), Standard Edition, was used on both of the applications in the experimentation test bed [Microsoft93]. The Standard Edition is the entry-level version of the product (the Professional Edition provides a larger suite of tools and controls).

Visual Basic uses "forms" as its primary layout and organization metaphor. That is, a form module is both the visual manifestation of the Graphic User Interface window and its components, and a programmatic artifact which the developer can access to establish the look and feel and the behavior of the application. One form is established as the master and is opened whenever the resulting program is executed. Each additional window, whether modal or modeless (referred to as a Multiple-Document Interface, or MDI, child), is manifested as a form. Forms have properties which are set by the developer and which can be altered programmatically during execution. Forms also may have behaviors which respond to pre-defined user-generated or system events, such as "Load," "Click," "Unload," etc. Each form is stored as a file with an extension of ".FRM".

The visual elements are referred to as controls. Common examples include push buttons, radio buttons, text boxes, combo and pop-up menus, static text and graphics. Each control type is shown as an icon on a toolbox window such that an instance of the control can be dragged from the toolbox and dropped onto a previously painted form. Once on the form, the control can be sized by either grabbing it with the mouse or by setting its left, top, height and width properties to the desired values. Other properties may be set, including aesthetics (e.g., colors, borders and fonts) and behaviors (e.g., whether visible or what happens to the mouse icon whenever it is over the control). Some controls include a special "Tag" property which can be used by the developer to add a customized attribute when needed.

Like forms, controls also may have behaviors which respond to pre-defined user-generated or system events, such as "Change," "Click," "GotFocus," etc.

Visual Basic also provides a suite of tools. There is a Menu Design tool for setting up the menu bar and accelerator keys. There is a Data Manager for creating data bases (in Access). There is a Setup Wizard for creating distribution disks for stand-alone executables.

As touched on above, Visual Basic is an event-based environment. Events can be spawned by the user, for example, by moving or clicking the mouse or by keyboard actions. The system can also programmatically generate events, such as when a control changes value or it loses focus. Events can also be used as a way of sending a message to the degree that one can include in code the ability to generate any registered event (including ones normally reserved for users). The event handler also keeps track of which object spawned the event and, if the control was part of a control array (a group of identical instances of the same control and its options), it passes the index to the calling member of the array. This allows the receiver of the message to alter its response according to who was the sender.

The "methods" themselves (i.e., the receivers of the event messages) are in fact subroutines written in BASIC, associated with a control and attached to the form which holds it. Each control has its own built-in set of events to which it will respond (advanced developers can also create and register their own additional ones). If the design calls for a certain behavior given a certain event, then that behavior is implemented as a BASIC subroutine whose name is the object name joined to the event name by an underscore (e.g., Sub btnQuit_Click). If the control is implemented as an array, then the subroutine would also set up the indexing (e.g., Sub txtCell_Change (Index As Integer)). However, this indexing scheme stops short of allowing the code to be self-aware (the concept of the "SELF" keyword will be discussed in Section 4.4).

The features and functionality of Visual Basic can be significantly extended by buying add-on widgets from Microsoft or third-parties and by writing your own custom controls (so-called VBX's) in C. There is currently a booming market for add-on custom controls for such diverse areas as improved GUI widgets, communication utilities, and image viewing tools.

To build an application in Visual Basic, start by creating and naming an empty form. Then set its prcperties, such as whether it will be sizable by users and the caption that should be displayed in the title bar. Next, drag, drop, name, and size each control and set its properties, such as whether it is initially enabled or whether it is reachable by user "tabbing."

For data base applications, special data controls must be set up for each data base connection. The data control enables the application to navigate, filter, and manipulate the attached data base, using SQL-like commands (e.g., WHERE, GROUP BY, etc.) supplemented by a set of API-like methods (e.g., MoveLast, MoveNext, etc.). The data control also enables certain visual controls, such as Text Boxes, to be directly linked to the data base such that their current value is automatically updated when the associated data base value changes and vice versa.

Next, for each relevant event of each form and control, add GUI-specific code. When the developer double-clicks on the object, the source code editor is opened with a default event subroutine already sketched in (e.g., _Change for Text Boxes and _Click for Buttons). The application-specific code is then added there, if that is the intended event, otherwise, a pull-down menu can be accessed to bring up any other event that is meaningful to that object.

Any general purpose code (i.e., routines that are not tied to the behavior of specific user interface control objects) is implemented in a BASIC module associated with the application under development (".BAS" file). Whether in a ".FRM" or a ".BAS" file, the general syntax, variable definition/scoping and data structures of the BASIC language apply.

4.3 Object Vision

Borland's ObjectVision (version. 2.1) was used as the second implementation vehicle for the data base application of the experimentation test bed [Borland91A]. Like Visual Basic, ObjectVision uses the "form" as the metaphor for the GUI. Conceptually, the construction of the forms and the visual elements contained on them, referred to as objects, is very similar to that described in Section 4.2. Unlike Visual Basic, ObjectVision has only one file for the entire application, stored with the extension ".OBV". Forms are used to implement Windows and Dialog Boxes, and are individually named. ObjectVision also provides support for creating menus and accelerator keys and runtime deployment.

The visual objects are quite similar in form and function to those discussed for Visual Basic. However, several of them are more powerful (i.e., require less programming to accomplish the same goal). For example, in Visual Basic, one must programmatically populate the contents of a pop-up menu; in ObjectVision, the developer can provide the contents, or the application can dynamically populate the menu itself based on data base values at runtime. Another more advanced feature is support for data restriction via the concept of content "pictures" (e.g., a local phone number would have a picture of ###-####).

ObjectVision's capability for linking to data bases is superior. The "links" tool allows one to create and modify data base schema in several popular PC-based formats, including Xbase, Paradox, B-trieve and even comma-delimited ASCII flat files. More importantly, the links tool examines objects on the forms that have built up to that point and attempts to match them up to the data base. Likewise, in data base creation mode, the schema is created based on the data types and lengths found on the forms.

Another data base related area where ObjectVision excels is in managing relational designs. Even though they are manifested in traditionally non-relational data base formats, ObjectVision itself establishes and maintains the contents as relational tables. This includes understanding of foreign keys and automation of developer-selected referential integrity rules. This tool also recognizes likely joins and automatically establishes them.

ObjectVision is also an event-driven tool, responding to both user-generated and system-generated events. Events can also be invoked programmatically, but such usage is less powerful since they cannot contain arguments or report which object spawned them.

The primary means of implementing application behaviors is by creating logic trees. These trees are constructed in a visual editor that provides developmental support, such as menus of available objects and guidance on legal maneuvers. Such trees are the closest equivalent to methods available in ObjectVision, and are one of two types. "Value Trees" are used primarily to provide derived data values (when the visual object to which it is attached remains null) and to perform data validation (when the visual object to which it is attached receives a value from the user). "Event Trees" are used for performing more general event-based duties and are created for specific events for specific object. For example, the "Quit" button object would likely have an Event Tree attached to it to instruct the application on how to respond to a "Click" event. The action to be taken when a leaf of the tree is reached is articulated using Lotus 1-2-3-like scripting language.

The events and script verbs can be extended by writing your own custom DLL's in C and registering them with the application at start-up.

If used correctly, and if the target application is data base-oriented, ObjectVision can be a powerful tool. Otherwise, it can be a drain to productivity. The first task is to construct the GUI. The steps required include: create and name an empty form; set its properties; drag, drop, name, and size objects; set their properties.

The second task is to add the behavioral dimension to the application by constructing the Event Trees and Value Trees for each relevant event of each form and control.

The final task is to use the "link" tool to attach the GUI objects to their counterparts in the data base. If data elements in the GUI are missing from the data base, the links tool will help create them. It will also prompt for referential integrity options and other helpful options, such as filters.

4.4 Smart Elements

Smart Elements (version 2.0), by Neuron Data, was used as the second implementation vehicle for the algorithmic application of the experimentation test bed [NeuronData94C]. Smart Elements is actually a package that integrates two independent Neuron Data products that had achieved success in the marketplace: Open Interface (version 3.0) and Nexpert Object (version 3.0). The Open Interface element can be used alone to build portable GUI front ends to C-based applications. The Nexpert Object element can be used alone to build portable Knowledge-Based Systems or object-oriented applications. Together, they form a very complete, graphical environment for building advanced, production-grade applications. Open Interface supplies the front-end processing and Nexpert Object provides the back-end processing.

On the surface, Smart Elements provides much the same set of capabilities and features as were discussed in the previous two sections. Using the "Open Editor" facility, windows and dialogs (modal or modeless) and the visual elements contained on them (referred to as "widgets") are created by dragging, dropping, sizing, and naming them and then setting their properties. The results are stored in two formats: a platform-independent ASCII resource file (".RC") and a platform-independent binary data base file (".DAT"). Push buttons, radio buttons, text edit boxes, combo and pull-down menus, text and graphics are just some of the standard widgets provided. Advanced users can extend or alter these widgets as they see fit.

Open Interface provides tools for setting up menu bar and accelerator keys and linking to knowledge bases (in Nexpert Object). There is also a C source code generator that creates a compilable version of the application front end.

Open Interface is highly event-driven in its architecture, while the behavior of Nexpert Object is governed by a complex and highly configurable agenda mechanism and inference engine which handles both events and chaining of rules. Behavior of the GUI is governed by procedures written either in the Open Interface scripting language or in C. The scripting language is itself based on C, and C programmers will find it a familiar environment to work in. On the Nexpert Object side, behaviors are implemented either by methods, using the Method Editor, or by rules, using the Rule Editor. Both elements support creation and inheritance of very generic code which can adapt itself to the situation at run-time, including interpretation of reserved words: "@"V for knowledge base atom names, "SELF" to represent the specific widget to which a script is attached, and "EVENT" to represent the event currently being processed.

Smart Elements is fully object-oriented in both its internal design and its use by developers. Both elements allow creation and exploitation of class/subclass/object(instance) hierarchies, including multiple inheritance. In Open Interface, the class structures can be displayed graphically in the Resource Browser and instances are created and modified in Open Editor; however, creation of new classes requires significant programming. On the other hand, in Nexpert Object, objects, properties, methods and rules are all written and modified by filling out dialog boxes. Results can be visualized in a series of graphical browsers, including a Rule Network and an Object Network. The notation used for these browsers is shown in Figure 7.

The classes, objects, properties, rules and methods of a Nexpert Object application are stored as a platform-independent ASCII knowledge base file (".KB"). The knowledge base can also be stored in a platform-specific compiled (binary) format.

Another unique feature of Smart Elements is that the environment is very open. For example, features can be extended by writing your own procedures or DLL's in C. More importantly, application modules built in Smart

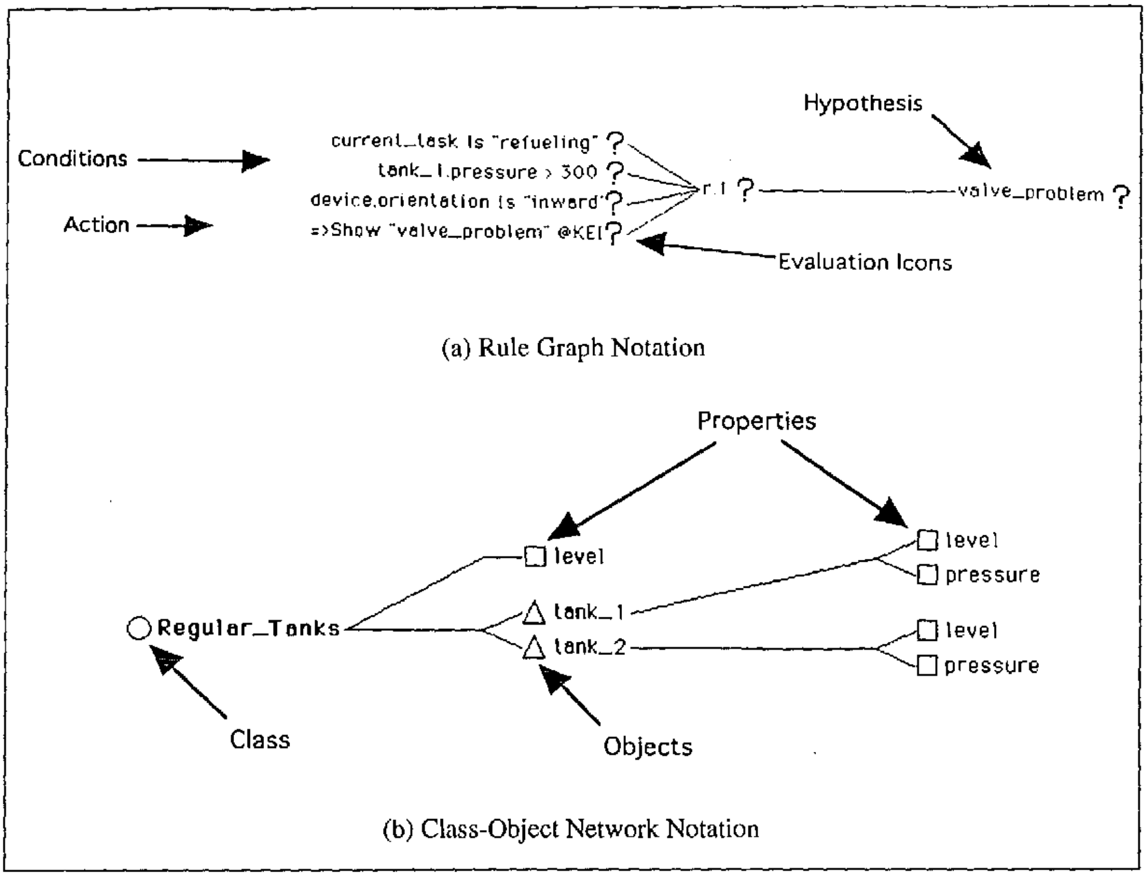


Figure 7: Smart Elements Rule and Object Notations [NeuronData94C, pp. 50-51]

Elements can be embedded into other C applications using a very robust Application Programming Interface (API).

One of the most powerful features of Nexpert Object is built in support for pattern matching in rules and methods. Pattern matching allows very generic code to perform searches of properties over class/object hierarchies and return a list of the matching objects. The list persists for the duration of the current operation and can thus be used with subsequent conditionals or actions of that operation. The list can also be passed as an argument to another operation. The value of this feature will become obvious in Section 5.2.3.

To build an application, first start up the Smart Elements Main Window. From here, one may either proceed to create the GUI in Open Interface or the back end in Nexpert Object. The tool is so robust, that there are numerous avenues to accomplish any task, so this treatment suggests merely one, starting with building the GUI. In reality, this is an interactive process, cycling among GUI widget editing, widget script writing, and editing of classes, objects, properties, methods, and rules.

First, display the Resource Browser, navigate to the "Win" resource, and double-click on it to start the Open Editor with an empty window. Use the tools, palettes and dialogs to create and name the required windows; set their properties; drag, drop, name, and size widgets; and set their properties. For each relevant event of each window, dialog and widget, add the GUI-specific script which will achieve the desired behavior (or alternatively, generate a C-source template and add C-code to implement behavior).

For knowledge-based applications, set up links between the GUI objects in Open Interface and knowledge base objects in

Nexpert Object. For data base applications, one can also set up data links, using either an add-on product named Data Elements for complex, client/server applications, or the built-in scripting calls for simple ones.

Now, implement the structural aspects of the application in Nexpert Object (i.e., classes, objects and properties) by opening and using the Class Editor, Object Editor and Property Editor, respectively. The results of this hierarchical construct can be visualized using the Object Network browser. Finally, implement the (non-GUI) behavioral aspects of the application in Nexpert Object (i.e., methods and rules) by opening and using the Method Editor and Rule Editor, respectively.

4.5 Layout

Layout, by Objects, Inc., was originally slated as the tool to be used to build the second algorithmic application in the experimentation test bed [Layout92]. Although it was ultimately abandoned for use in the test bed, it does provide an interesting insight into visual programming. In particular, a recursive version of the classic factorial algorithm was visually programmed in Layout as part of the evaluation process. Although somewhat tangential to the main thrust of this thesis, the purpose of this section is to demonstrate how a such an abstract concept as recursion can be programmed visually.

The programming metaphor used by Layout is the flow chart. Functional code modules are represented by icons known as "Black Boxes." Cascading palettes of Black Boxes are available from which one may select and drop onto the flow chart. Black Boxes can have inputs and/or outputs, can return a value to the Black Box which called it, and can receive and send messages with arguments. Flow Charts can be broken up into callable Procedures which themselves can take arguments and return values. Woven throughout the Flow Chart and Black Box metaphor are supporting ones, such as "Filling out a Card" for opening a user dialog window.

Layout has a GUI painting tool to develop user windows and populate it with the (now) usual slate of widgets. These widgets are connected to the Flow Chart and its Black Boxes in a very restrictive fashion. (ASIDE: The lack of flexibility in the connection between the GUI and the Flow Chart became the downfall of this tool for the Tic Tac Toe game. Many hours of work-around attempts, discussions and faxes with Layout Tech Support yielded no viable solution.) User input data is collected and organized using the data card (or index card) metaphor. There is also a tool for creating/managing the variables used in the Black Boxes.

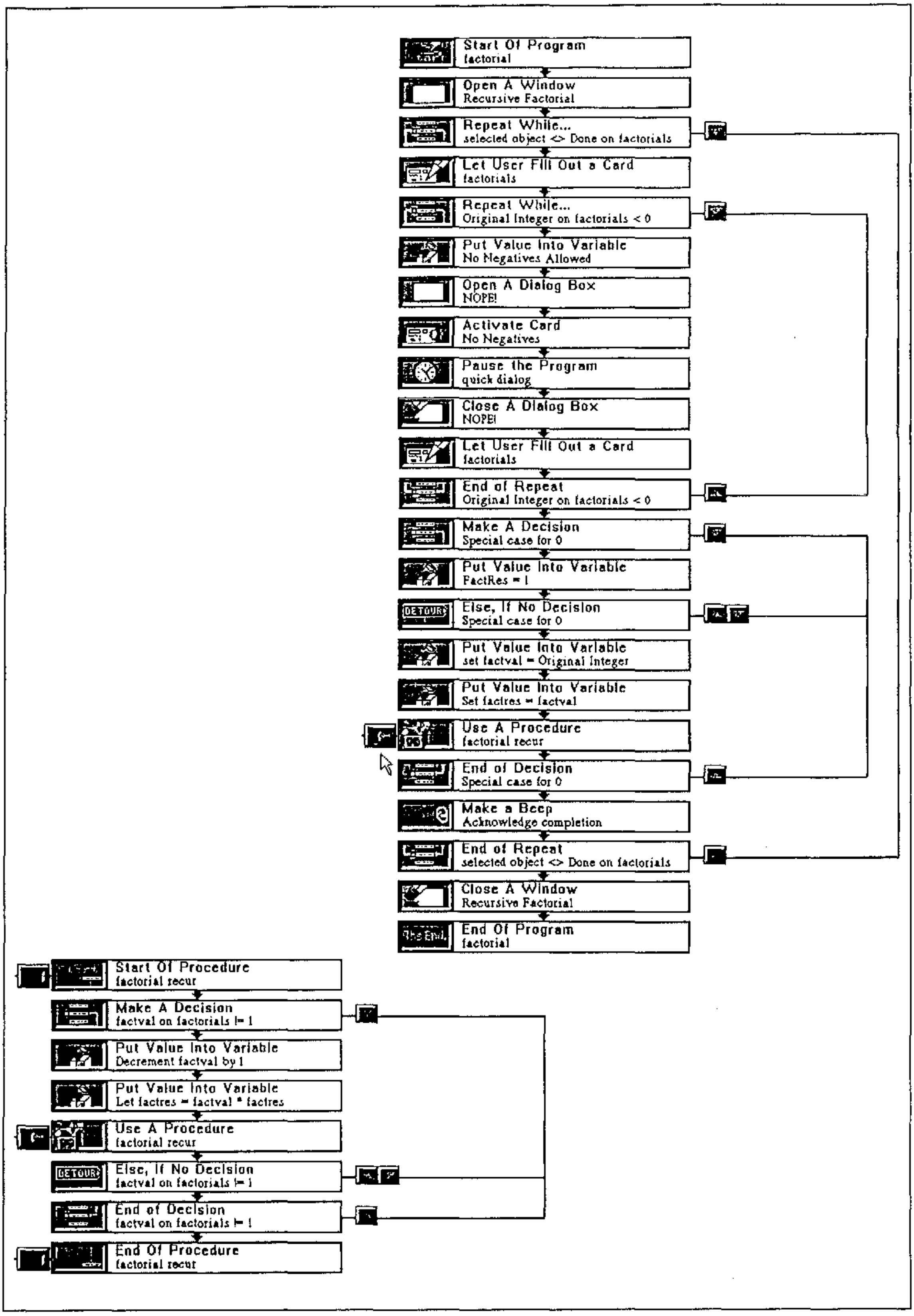
To build an application in Layout, use the Flow Chart tool to begin visually building the application. When a Black Box is chosen that requires a window, a screen painting tool will open which will allow you to create and name an empty window (or card) and set its properties. You may also drag, drop, name, and size objects and set their properties. When a Black Box is chosen that requires an equation with variables, a Variables tool will open which will allow you to create the needed data elements. The process of editing the Flow Chart and its Black Boxes, including the filling out of associated dialogs, when requested, is repeated until the desired application behavior is achieved.

With that background, the concept of visual recursion can now be presented. Of course, the impact of visual programming is significantly diminished when reduced to a paper-based portrayal, such as this. However, the point that visual programming opens new software engineering opportunities and challenges should not be lost.

The factorial application presented here provides a simple graphic user interface to accept the input value and display the output value. The application also provides input validation and feedback to the user if the input is unacceptable (i.e., not a positive integer). Last, but certainly not least, it must calculate the factorial of the input value and place the result in the output value. It is this last step that is implemented recursively. The pseudo-code for a recursive version of the factorial algorithm is as follows ("input" and "output" are global integers):

```
factorial (input, output)
     If input > 1 Then
          input = input - 1
          output = output * input
          factorial (input, output)
     End If
End function
```

Figure 8 shows the Layout Flow Charts for the Main ("factorial") and Recursive Procedures ("factorial recur"). The main program sets up the user window; sets up the main event loop (while the user has not selected the "Done" button); lets the user enter an input value (in an data entry object named "Original Integer" on a Card named "factorial"); sets up entry validation and correction loop (while user input < 0); handles the special case for 0! (If user input = 0, result = 1) and otherwise initializes the input ("FactVal") and output ("FactRes") variables and calls the recursive procedure (the Else portion); makes a



```
Start Of Program
factorial
Open A Window
Recursive Factorial
Repeat While...
selected object <> Done on factorials
Let User Fill Out a Card
factorials
Repeat While...
Original Integer on factorials < 0
Put Value Into Variable
No Negatives Allowed
Open A Dialog Box
NOPE!
Activate Card
No Negatives
Pause the Program
quick dialog
Close A Dialog Box
NOPE!
Let User Fill Out a Card
factorials
End of Repeat
Original Integer on factorials < 0
Make A Decision
Special case for 0
Put Value Into Variable
FactRes = 1
Else, If No Decision
Special case for 0
Put Value Into Variable
set factval = Original Integer
Put Value Into Variable
Set factres = factval
Use A Procedure
factorial recur
End of Decision
Special case for 0
Make a Beep
Acknowledge completion
End of Repeat
selected object <> Done on factorials
Close A Window
Recursive Factorial
End Of Program
factorial
```

```
Start Of Procedure
factorial recur
Make A Decision
factval on factorials != 1
Put Value Into Variable
Decrement factval by 1
Put Value Into Variable
Let factres = factval * factres
Use A Procedure
factorial recur
Else, If No Decision
factval on factorials != 1
End of Decision
factval on factorials != 1
End Of Procedure
factorial recur
```

Figure 8: Layout Flow Charts of Recursive Factorial

"beep" to acknowledge completion; closes the window and exits (once the user has clicked on "Done").

The recursive procedure tests the input value ("FactVal") to verify that it has not yet become decremented all the way to 1 (if it does equal 1, the procedure is immediately exited); if not equal to 1, the procedure decrements the input to the procedure by 1, sets the result ("FactRes") equal to the old result times the newly decremented input value and calls the recursive procedure; when control returns, the procedure is exited.

The concept of a visually programmed recursive function may be hard to imagine. However, when the Flow Chart is displayed, it only shows the current level of recursion. One can drill down to the next level in the recursion by clicking on the small icon to the left of the "Use a Procedure" Black Box and come back up back by clicking on the small icon to the left of the "Start of Procedure" Black Box. The developer may drill down as many levels as desired and Layout will keep repainting the same recursive procedure; however, it is keeping track of the levels because the icon for coming back out will have to clicked just as many times to get back to the main procedure.

Chapter 5

Experimentation Test Bed

The experimentation test bed described in this chapter was conceived to create a controlled environment in which to observe the interactions and interdependencies among the software engineering methodologies discussed in Chapter 3 and the visual programming languages and tools discussed in Chapter 4. The observation capture mechanism discussed in Chapter 2 was applied throughout the design and implementation process as noteworthy synergies and conflicts were encountered.

5.1 Customer Support Tracking System

The Customer Support Tracking System (CSTS) was inspired by a project ongoing at the author's workplace for which the client was wanting the company to take incoming technical support calls for a fielded software system and then bill them for the time used. CSTS was conceived as a relational data base aimed at managing a group of users of a fielded software product. CSTS was designed using Gane & Sarson DFDs and ERDs using System Architect and implemented in Visual Basic and ObjectVision. The system must maintain a data base of licensed users, including relevant data about that user and the company for which he or she works. It must also track the duration and content of technical support calls taken from licensees. The GUI must serve as both the end-user and maintenance interface.

5.1.1 CSTS Design

The CSTS Design Package is provided as Appendix A. The package consists of a System Requirements Definition statement, the Gane & Sarson DFDs and associated Process Definitions, the ERD and associated Data Definitions.

The design was created using the System Architect CASE tool. The tool is not only responsible for producing high-quality printouts of the various diagrams and definitions, but also maintains an association between the symbols on the diagrams and the contents of the definitions data base. Thus, System Architect was not used simply as a drawing tool, but rather, was used as a repository of design information, both symbolic and definitive. It also helped reveal the need to normalize the data. The original requirements did not specify a separate table for the licensee (person) and the company for which they worked. It turns out that one company may have multiple licensees and as such the company data ought to be stored and maintained separately. Therefore, the design divided the CUSTOMER table that was specified into separate LICENSEE and COMPANY tables.

5.1.2 CSTS Implementation in Visual Basic

The listings and screens for the Visual Basic version of CSTS are provided as Appendix B. There are four sections in the appendix, one for each of the major modules. Each module consists of a printout of the GUI screen, the definitions of that screen and the visual objects it contains, and a listing of the Visual Basic source code for the object behaviors. The main module, CSTSMain, provides the "Customer Support Tracking System" window and is saved under the name CSTS.FRM. The company information maintenance module, ChangeCompany, provides the "Change Company Information" window and is saved under the name CHANGECO.FRM. The technical support module, SupportCall, provides the "Support Calls" window and is saved under the name SUPPORTC.FRM. A module for deleting companies, CSTSMnt, provides the "CSTS Company Maintenance" window and is saved under the name CSTSMNT.FRM. This application has no generic (.BAS) module files.

To use the Visual Basic version of CSTS:

- Start Visual Basic

- Load the CSTS Project (CSTS.MAK)
- Run the application and the CSTS main window will open
- You may exit the application either by first selecting the "File" item from the menu bar or by double-clicking on the system icon (the bar in the upper left corner of the window)
- You may browse existing licensees using the navigation icons (they are similar to VCR buttons) to go to the first, previous, next and last record.
- You may edit the licensee information (the company information cannot be edited from this screen)
- You may type a value in the Licensee ID box; if that ID already exists, that record will be displayed, otherwise, a new licensee record will be created and the "Change Company Information" window will be opened (since every licensee must have a company associated with it)
- You may Click on the "Change Company Info" Button to open the "Change Company Information" window
- On the "Change Company Information" window, you may edit the data, browse the existing company records (using the navigation icons), select an existing company record directly (using the pull-down menu) or create a new company record by typing in a new name; you may either "Accept" the changes or "Cancel" them, either of which will return you to the main CSTS screen

- From the main CSTS screen, you may Click on the "Support Calls" button to open the "Support Calls" window
- On the "Support Calls" window, you may type in a description of the call of any length into the scrollable text box; you may browse the past support calls to that licensee using the navigation icons; you may either "Accept" the support call information or "Cancel" it, either of which will return you to the main CSTS screen
- From the main CSTS screen, you may delete a company record or a licensee record by first selecting "Maintenance" item from the menu bar (NOTE: cascading deletion of the related data is not automatically provided by Visual Basic and was not programmatically implemented)

During the implementation, several interesting things occurred. One positive experience had to do with a powerful data base connection device built in to Visual Basic (known as the "data control"). It allows visual objects to be directly linked to fields in a data table and also automatically provides the data base browser feature.

A disappointing experience was discovering that the pull down menus (e.g., combo boxes) are not populated automatically at run time, but rather must be populated programmatically. Other tools, such as ObjectVision, are able to scan the contents of the data base at run time and populate the menu without the need to write a single line of code.

5.1.3 CSTS Implementation in ObjectVision

The listings and screens for the ObjectVision version of CSTS are provided as Appendix C. There are three sections, one containing printouts of the three screens, one containing printouts of the behavioral event trees and scripts, and one containing sample screen shots from the ObjectVision development environment. The three screens are: the "Customer Support Tracking System" window, the "Change Company Information" window, and the "Support Call" window. Since this application was constructed in a visual programming tool, it has no source code listing, per se. Instead, event tree diagrams with their "mini-scripts" are provided for: the "Change Licensee ID" (text edit) Event, the "Open Change Company Information" (form) Event, the "Click New Company" (button) Event, the "Click Return" (button) Event (on "Change Company Information"), the "Click Cancel" (button) Event (on "Change Company Information"), the "Open Support Call" (form) Event, the "Click Accept" (button) Event (on "Support Call"), and the "Click Cancel" (button) Event (on "Support Call"). The screen shots portraying typical parameters that were set for various dimensions of the application are: combo box setup (attribute menu, Field Type selection, Expected List dialog for combo box showing "Automatic"), date field setup (attribute menu, Field Type selection, Date Type selection dialog), non-editable text edit box setup (attribute menu, Protection choices dialog), constrained data entry text box setup (attribute menu, , Field Type selection, Picture String dialog), data base table creation (Data Links dialog, Paradox Link Creation dialog, Data Base Table Creation dialog), and data base filter setup (Data Links dialog, Paradox Link Creation dialog, Optional Link Capabilities dialog, Link Filters dialog).

To use the ObjectVision version of CSTS:

- Start ObjectVision
- File|Open CSTS.OVD (the CSTS main window will open)
- You may exit the application either by double-clicking on the system icon (the bar in the upper left corner of the window)
- You may browse existing licensees using the navigation icons (they are similar to VCR buttons) to go to the first, previous, next and last record.
- You may edit the licensee information (the company information cannot be edited from this screen)

- You may select an existing licensee record directly from the Licensee ID box (using the pull-down menu)
- You may type a value in the Licensee ID box; if that ID already exists, that record will be displayed, otherwise, a new licensee record will be created and the "Change Company Information" window will be opened (since every licensee must have a company associated with it)
- You may Click on the "Change Company Info" Button to open the "Change Company Information" window
- On the "Change Company Information" window, you may edit the data, browse the existing company records (using the navigation icons), create a new company record by clicking "New Company" and typing in the new name, or deleting the current company record by clicking on "Delete"; you may either accept the changes by clicking "Return" or "Cancel" them, either of which will return you to the main CSTS screen
- From the main CSTS screen, you may Click on the "Support Call" button to open the "Support Call" window
- On the "Support Call" window, you may type in a description of the call of any length into the scrollable text box; you may browse the past support calls to that licensee using the navigation icons; you may either "Accept" the support call information or

"Cancel" it, either of which will return you to the main CSTS screen

- From the main CSTS screen, you may delete a licensee record by clicking on "Axe".

The primary negative impression during the development process was the nearly impossible task of documenting the object attribute selections. There is no direct way to do this. Taking screen shots (as was done for the few samples in Appendix C) would be prohibitively time consuming on even a medium-size project.

On the positive side, the data base connection (link tool) was most impressive. ObjectVision successfully generated "straw man" data tables based on visual objects, automatically setup referential integrity constraints, automatically setup joins at run time, and automatically populated the combo box menus. At first, the fact that the System Architect Schema Generator tool did not offer support for any of the data base formats supported was disappointing. However, the schemas for all three data tables were ghosted in based on the GUI objects and polished up in a matter of minutes.

5.2 Tic Tac Toe

The Tic-Tac-Toe game was inspired by an assignment in the Software Tools class. It provides at straight-forward application that is both visual and does not entail a data base. It does, however, require (simple) logical and numeric algorithms. Tic Tac Toe was designed using the Coad/Yourdon methodology and implemented in Visual Basic and Smart Elements.

5.2.1 Tic Tac Toe Design

The Tic Tac Toe Design Package is provided as Appendix D. The package consists of a System Requirements Definition Statement, the Coad/Yourdon Analysis and Design step results, and the Coad/Yourdon Object State Transition diagrams.

The design was not created using a CASE tool, but rather was performed and documented manually. The analysis and design process was nonetheless a productive exercise. The information spelled out during completion of the OOA/D steps did indeed lay out a course for direct implementation. The mapping from the design to the programming environment was considerably better for Smart Elements than for Visual Basic, as would be expected considering that the former is a fully object-oriented tool.

The original design did not include the Object State Transition Diagrams. During the Visual Basic implementation, the need to understand the cascading sequence of events became evident (the author kept getting lost in a not-so-visible web of events causing side-effects causing more events and so on). At that point, effort was directed back to the design phase and the states and the events which could cause a change in state were laid out in a manageable and understandable fashion.

The design calls for nine identical cell objects to represent the nine locations of play on a Tic Tac Toe board. There are also eight conceptual summation objects to represent the state of the three rows, three columns and two diagonals which are possible on a Tic Tac Toe board. The object representation scheme is rounded out by a game state object that contains the current state of play.

The key to the algorithms used by the gaming engine is to represent the player's "X" with the integer "1", the game's "O" with the integer "-1", and an unused cell with "0". This allows the state of each row, column and diagonal to be unambiguously discerned by simply adding up the values of its three cells: +3 means the user has won, -2 means the game will take the (necessarily) remaining empty space for the win, and +2 means the game will take the (necessarily) remaining empty space for the block. The sums are also used to help discern several special cases having to do with the user attempting to get the game into a double-bind (where there are two rows/columns/diagonals that must be blocked, referred to as a "wedge").

The strategy devised for the gaming engine is, in order of priority:

1. Test for user win (any sum = +3) ==> Game Over

2. Test for game win (any sum = -2) ==> Find and Pick Empty Cell and Game Over

3. Test for block (any sum = +2) ==> Find and Pick Empty Cell

4. Test for diagonal wedge (sum of the sums of both diagonals = 0) ==> Pick Center or Top Edge

5. Test for edge wedge (the only one of concern is when the sum of the sums of the bottom row and the right column = 2) ==> Pick the Lower Right Corner Cell

6. Pick the next empty cell according to the following "search pattern":

2	6	3
7	1	8
4	9	5

5.2.2 Tic Tac Toe Implementation in Visual Basic

The listings and screens for the Visual Basic version of Tic Tac Toe are provided as Appendix E. There are four parts in the appendix, one for each type of information. The first part is simply the GUI screen for the application. The second part contains the definitions of that screen and the visual objects it contains (saved as MAIN.FRM). The third part provides the a listing of the Visual Basic source code for the object behaviors (also in MAIN.FRM). The fourth part is the Visual Basic source code for the generic functions used by the application (saved as TTT.BAS)

To use the Visual Basic version of Tic Tac Toe:

- Start Visual Basic
- Load the Tic Tac Toe Project (TTT.MAK)
- Run the application and the Tic Tac Toe window will open

- You may exit the application at any time either by clicking on the "Quit" button or by double-clicking on the system icon (the bar in the upper left corner of the window)
- You may click on any cell to make your first move, or Click on the "You Go First" button if you want the game to make the first move
- You will always be "X" and the game will always be "O"
- Notice that the mouse cursor icon changes whenever it is placed over a mouse-sensitive region of the game board.
- After each move by the game, you may select another empty cell for your move
- Notice that the game will not allow you to make two simultaneous moves or put your "X" in an already occupied cell
- The game will notify you of the final result when there is "three in a row" or when there are no more moves
- Warning: the game cannot be defeated, so do not spend more than two or three hours trying

Since design of the Tic Tac Toe game required both a symbolic and numeric representation of each cell on the board, the "Tag" property of Visual Basic TextBox object was put into service. Whenever the Text value of a cell changed to an "X", a "_change" method was invoked which set the Tag value of that cell to +1. Likewise, when the game decided its move, it set the Text value to "O" and let the change method set the Tag value to -1. This technique allowed the human to see the symbolic values and the game to apply its algorithms to the numeric values of the same objects.

The Visual Basic's capability to allow creation of an indexed array of identical visual objects was exploited in the implementation. Specifically, the "object_Change" method (mentioned earlier) was implemented as a completely generic subroutine. This routine was then attached to the first cell that was created for the game board. The other eight cells were added as indexed "clones" of the first one. In this fashion, all nine of the individual (indexed) text cells on the game board shared the same (identical) subroutine for handling changes in the value of a text cell. Thus, the code fragment shown below was written once and reused eight times with no additional programming required:

```
Sub txtCell_Change (Index As Integer)
     If txtCell(Index).Text = "X" Then
          txtCell(Index).Tag = 1
          ElseIf txtCell(Index).Text = "O" Then
               txtCell(Index).Tag = -1
          End If
```

```
        End If
    End Sub
```

5.2.3 Tic Tac Toe Implementation in Smart Elements

The listings and screens for the Smart Elements version of Tic Tac Toe are provided as Appendix F. There are four parts in the appendix, one for each type of information. The first part is simply a printout of the GUI screen for the application. The second part contains the Open Interface definitions of that screen and the visual objects it contains (saved as TTT.RC) Since the Open Interface scripts are formatted in an unfriendly manner in the ".RC" file, a manually edited listing of the scripts is also included. The third part presents the Nexpert Object "source code" for the Knowledge Representation Scheme (i.e., the Classes, Objects and Properties and how they are connected to one another). This part is a combination of screen printouts from the browsers and editors of the development environment plus a textual listing of the Knowledge Base (saved as TTT.KB). The fourth part is the Nexpert Object "source code" for the rules and methods used to provide the behavior of the application. Once again, this part is a combination of screen printouts from the browsers and editors of the development environment plus a textual listing.

To use the Smart Elements version of Tic Tac Toe:

- Start Smart Elements (this requires a security key)
- Load the Tic Tac Toe Knowledge Base (TTT.KB); this establishes the gaming engine in Nexpert Object
- Load the Tic Tac Toe compiled resource file (TTT.DAT); this establishes the GUI engine in Open Interface
- Navigate to the Open Editor main dialog (first, display the Resource Browser, navigate to the "Win" resource, and double-click on it to start Open Editor, which will display the Tic Tac Toe window in development mode
- Run the application by clicking on the "Test" button
- You may exit the application at any time either by clicking on the "Quit" button or by double-clicking on the system icon (the bar in the upper left corner of the window)
- You may click on any cell to make your first move, or Click on the "You Go First" button if you want the game to make the first move
- You will always be "X" and the game will always be "O"
- Notice that the mouse cursor icon changes whenever it is placed over a mouse-sensitive region of the game board.
- After each move by the game, you may select another empty cell for your move
- Notice that the game will not allow you to make two simultaneous moves or put your "X" in an occupied cell

- The game will notify you of the final result when there is "three in a row" or when there are no more moves
- The current state of the game is displayed for information purposes and to give you something to do besides fume over the fact that the game cannot be defeated

Implementing Tic Tac Toe in Smart Elements was a truly enjoyable experience. Although several dead ends and need for work-arounds were encountered, all in all, the environment delivered on the productivity and visualization benefits touted for a high-end visual programming tool. There was perhaps a factor of more than 5 times on productivity over the Visual Basic implementation.

Incremental development in an object oriented tool environment also paid rewards. The most dramatic example was development of the part of the gaming engine that manages the state of the game board (i.e., the sums of the eight rows, columns and diagonals). Once the prototype (one row of game board) was operating correctly, it took less than 15 minutes to create a fully operation game board. This rapid scalability was due to the self-maintaining quality of the board objects achieved through inheritance of generic methods. All that was required was to clone the other six cells, clone the other seven "summation" objects and connect the appropriate three cells to the corresponding summation object. It worked on the very first try, requiring not one line of new code and not one session of debugging.

The concept of Pattern Matching in Nexpert Object was used extensively in the generic methods mentioned above. For example, the conditional (IF clause) of one of the rules was:

```
(= (<|Sums|>.Sum) 2)
```

This says "If the .Sum property of any child of the Sums Class equals 2 (i.e., two "X"'s), then put the name of that object in a list." That list can then be used in subsequent operations in that rule and can be passed as an argument to a method, as seen in the action (THEN clause) of that same rule:

```
(SendMessage ("mthdPickLastCell")
     (@TO=|Cells|.Val;@ARG1=<|Sums|>;))
```

This says "Send a message to the .Val property of all children of the Cells class, and invoke the PickLastCell method, with the previously generated list, locally known as <|Sums|>, as an argument." The method, expecting as an argument a list of objects with one member whose .Val property is 0, then sets off to identify its name (remember that for a summation to have been equal to 2, two of the cells in that row, column or diagonal must have been filled with a 1, leading the knowledge that the third must still be empty). Note that the same method is used by the rule seeking a row, column, or diagonal equaling -2 (i.e., two "O"s) to identify the empty cell.

One of the challenges of implementing Tic Tac Toe in Smart Elements involved the need to mold the (tool-independent) design into a distributed processing architecture. This rethinking was required because the GUI engine (in Open Interface) and the gaming engine (in Nexpert Object) are completely independent processes communicating through a software bridge. Thus, the design had to be augmented to flesh which jobs should be done inside which process and how the necessary inputs and outputs of each process should be communicated to the other.

Chapter 6

System Architect to Visual Basic Bridge Prototype

One of the current trends (discussed in Section 7.2) has to do with the merging of CASE tools and VPTs. This trend embraces the concept of automatic programming. Indeed, System Architect today can automatically generate SQL data base schema from a data model built in the tool. However, many other implementation aspects remain a manual process in all but the most advanced tools. One such aspect is that of graphic screen design and implementation. System Architect provides a component for "painting" a Graphic User Interface screen and then automatically generating a generic MSWindows dialog file (which is characterized by a ".DLG" extension). Unfortunately, Visual Basic does not recognize such dialog files, but rather, uses its own file format for storing user interface data (characterized by a ".FRM" extension). Therefore, the author created a prototypical bridge program, named SA2VB.EXE, which automatically generates Visual Basic Forms (*.FRM) from screens (*.DLG) generated using System Architect. The purpose of this effort was to demonstrate that such bridges are possible, are practical and should be pursued as part of the maturation process of these tools.

6.1 SA2VB.EXE Design and Scope

To design a GUI translator, one must first understand the syntax, coordinate system, and naming conventions used on both sides of the translation (i.e., source and target). Several reference documents were digested in order to pin down these topics on the Windows Dialog side [Microsoft92] [Petzoid92] [SysArch94B]. On the Visual Basic side, these topics primarily were discovered by "reverse engineering" example Forms, with some help from the Programmer's Guide [Microsoft93]. For example, one can populate a form (window) with an assortment of controls (widgets) each having an assortment of options selected and then examine the ".FRM" file for that form to discover the representation scheme.

Because of the difference in coordinate systems between System Architect and Visual Basic, the size and location parameters had to be converted. The algorithms for doing this were applied as each parameter was handled. In Windows dialog files, x-coordinates and width parameters are based on 1/4 of an average character width while y-coordinates and height parameters are based on 1/8 of an average character height [Petzoid92]. For standard Windows GUI applications, a character is 8 units wide and 16 units high, thus making the coordinate system symmetrical in both axes. In Visual Basic, the default coordinate system uses "twips," which are defined in terms of size at 1440 twips per logical inch [Microsoft93, pp. 353-354]. Visual Basic on-line help, under "ScaleMode Property," further explains that a standard character is 120 twips wide and 240 twips tall. For most situations, the conversion algorithm was simply the parameter's dialog-value times 30, which is a good approximation for converting character height and width fractions to twips (i.e., 120 twips/char width ÷ 4 dialog units/char width or 240 twips/char height ÷ 8 dialog units/char height). The main window was the exception, requiring the height and width to be offset by an additional 360 and 60 twips, respectively, to account for a slight difference in representing the origin of the window.

Since there are an enormous number of controls and options, the scope of the translator was limited to the window itself and three fundamental control types: Text Edit Boxes, Pushbuttons and Static Text. The recognized options for the main window and these three controls are summarized in Table 3. Other design decisions were to implement the bridge as a DOS-based utility program using Borland Turbo C++, Version 1.01 [Borland91B], to accept the source file name as a command line argument and to output the resulting Visual Basic file with the name "out.frm".

Dialog Items	Recognized Options
Main Window	Size/location No Border Fixed Single Border Thick (sizable) Border Control Box (the menu box in upper left corner) Maximize Button Minimize Button Caption text
Text Edit Box	Size/location Default text Vertical Scroll Bar Horizontal Scroll Bar Both Scroll Bars Multiline Right Justified Left Justified Centered Border Box
Pushbutton	Size/location Caption text Default button
Label (Static Text)	Size/location Label text Right Justified Left Justified Centered

Table 3: SA2VB Scope Matrix

6.2 SA2VB.EXE Implementation and Testing

The source listings for SA2VB.EXE can be found in Appendix G. The program opens the source file (using fopen) and scans it word by word (using fscan) to identify and test each token. Because the possible tokens are well constrained, the program was written using statically defined variables (up to 256 characters) to represent each word. When an in-scope item is encountered, the program appends its Visual Basic equivalent to the output file (using fprint). The program is written such that it harmlessly ignores out-of-scope tokens.

The program first sets up the window and its options, and then recursively seeks out and handles the controls and their options. The coordinate algorithm discussed earlier is applied as each control is handled; however, a defined constant, FACTOR, is used in case a non-standard video configuration creates a need for a different conversion factor. When the end token is encountered, the program wraps up out.frm and closes out the input and output files.

The program was tested using several cases designed to exercise its various features. In addition, randomly selected System Architect screen files developed by various employees of PathTech Software Solutions, Inc., were converted to ensure that the program could handle "real world" conditions.

6.3 SA2VB.EXE Application

To use SA2VB.EXE, one must first go into System Architect and use its "Graphic Screen" module to define a user interface. This is done by dragging, dropping and shaping the window and its components and by filling in the details for each component in "behind the scenes" dialog boxes. These dialog boxes, called up by either double-clicking on the graphic component or by clicking the right mouse button on it, are where the various options are selected. Once the screen image and its properties are satisfactory, one must then invoke the System Architect "Generate Dialog" feature. This causes System Architect to automatically create a Windows standard compliant ".DLG" file containing the appropriate control parameters and definitions.

Once a valid ".DLG" file is available, one must shell out to DOS and execute SA2VB.EXE with the dialog file as a command line argument, as follows:

```
SA2VB TEST.DLG
```

Next, one must launch Visual Basic and add the file named "OUT.FRM" (this is found under the "File|Add File" menu). Finally, the newly created user interface can be displayed and examined (and later saved under a meaningful name) by double-clicking on "OUT.FRM" in the Project Window.

This process was carried out for numerous test files, as discussed earlier. One example, showing the image and file listing before and after conversion, is included as part of Appendix G.

Chapter 7

Conclusions

The conclusions which can be drawn from this research effort have been divided into four areas of discussion. The first points to the observations collected during construction of the test bed. The second examines the current trends in the literature as they relate to this effort. The third section offers a series of guidelines aimed at bridging the gap between the subject technologies. The last summarizes the results and findings of the overall effort.

7.1 Observation Results

As expected, a variety of instances occurred where subject technologies worked in concert or in conflict. The self observation forms which capture these instances are collected in Appendix H. In addition, three colleagues who regularly deal in the subject technologies were interviewed to verify that the specific findings based on the test bed could be reasonable generalized. The peer observation forms are collected in Appendix I.

7.2 Anticipated Trends and Developments

Lowry forecasts the emergence of "knowledge-based software engineering" where CASE tools will evolve to include semantic content and where "software engineers will be delivering the knowledge for generating software rather than the software itself"[Lowry92]. Although well out of this author's price range, Intellicorp's Object Management Workbench (OMW) is well on its way to fulfilling that forecast [Hanna94]. Based on a fully object-oriented methodology known as Martin-Odell, OMW allows software engineers to create analysis and design diagrams which are directly executable and from which C++ source can be generated and compiled when the time is right.

In early 1994, O'Brien expressed his concern that CASE tool vendors were not keeping up with the rapid adoption of event-based architecture, object-orientation, component-based development and points to the need for a new generation of CASE tools [O'Brien94]. If this is indeed the case, the market need for new tools will draw them out of the vendors, if not the current ones, then new ones who crop up to fill the void [Linthicum94, Constantine95]. One researcher, pondering how difficult it was to avoid methodology obsolescence, envisioned a marriage of formal software engineering techniques and visual programming which he called Visual Software Engineering [Chang94].

The forerunner of such a marriage is automatic program generation. This is a relatively mature technology for non-visual settings. However, automation of conceptual models, often best candidates for visualization, is in its infancy. Blum reinforces this belief as he classifies software engineering methodologies according to whether they are more concerned with conceptual modeling or formalization, and then points to a gap between the two [Blum94]. Indeed, it was the gap between GUI design and implementation that spawned the idea for the SA2VB bridge prototype that was discussed in Chapter 6. The (hopefully interim) need for such utilities was underscored by Keuffel as he described the techniques he used to narrow the gap between Evergreen's EasyCASE and Microsoft FoxPro [Keuffel94]. The off-loading of routine programming tasks to end-users via "Wizards" is another concept that could be carried into the software engineering domain (today's Wizards are targeted at helping office workers create custom charts, forms, layouts, etc.) and is a trend that bodes well for the proposed marriage [Kiyooka95].

From the visual programming point of view, tools with more and more power and flexibility are reaching the marketplace. The most visual, such as IBM's VisualAge, are empowered by full object orientation, an intuitive 4GL, and support for relational data base concepts. VisualAge, implemented in a combination of Smalltalk and itself (making it both a tool and a language), offers visual design and development of client/server applications, including SQL schema generation and application partitioning [Hanna94, Harding95].

The rapid growth in the use of visual programming tools is being driven by their ability to deliver reasonably transparent access to object-oriented programming and an easier transition to event-based architectures, GUI front ends, component-based development and the like [Jicha94, Schmidt95]. However, it is unlikely that even the best visual programming environment can achieve its full potential if it is not also delivering a sharable, understandable, reusable, printable, widely accepted software engineering methodology.

7.3 Guidelines for Development

This section of the thesis presents a collection of guidelines for finding synergy and avoiding conflicts between software engineering methodologies and visual programming tools and languages. The guidelines that follow are presented as a bridge to the day when CASE tools and visual development tools are truly one in the same. The guidelines are divided into the four major categories of User Interface, DB Schema, Event-Based and/or Object-Oriented Design, and Function Design.

7.3.1 User Interface

Look for CASE tools that can automatically generate the Graphic User Interface "code" in the native tongue of the Visual language/tool, thus avoiding duplication of effort.

If such a CASE tool is not available, or if a CASE tool is not being used on the project, then consider using the layout mode of the implementation language/tool as the design tool. Most Visual Programming languages and tools provide the capability to quickly sketch out a screen, including titles, labels, and graphics, data presentation and edit areas, and user control devices (e.g., pull-down menus and buttons).

Look beyond the obvious in stretching the features of the tool to make it meet the specifications. For example, the design may call for a text edit object with certain behaviors regarding clicks or changes, but the tool may not provide all of the desired behaviors for a text edit object. However, a 1 x 1 spreadsheet or grid object looks exactly like a text edit object and may provide enhanced behaviors needed in the design. This guidance comes with one caution, however. The benefit of stretching a tool may reach a point of diminishing return, leading to excessive labor costs and lost productivity.

Many languages/tools provide the ability to create custom classes, objects or widgets. Doing such would provide the ability to incorporate whatever "generic" attributes and behaviors the object should have (e.g., a BoardCell in the Tic-Tac-Toe application) and then create instances of it in the User Interface. If the development team includes strong computer science capability, then major extensions can sometimes be coded using the underlying language of the visual tool. For example, one can significantly alter the features and behavior of IBM's Visual Age using its underlying language, Smalltalk. Similarly, one can extend Visual Basic by writing their own so-called "custom controls" (identified by a ".VBX" extension) using C.

Establish a GUI object naming convention which expresses the type of object, whether it is native or derived and which options apply. Some object types are pervasive enough to now be considered generic, such as a text edit object or a combo box object. If the implementation tool is known, then the naming convention can be more explicit in how it represents the tool's objects and their options. An example is shown in Table 4.

GUI Object	Naming Pattern
Simple text edit	tedObjectName
Specialized (i.e., derived) text edit	projXtedObjectName
Delete Record Button	delbtnObjectName
Delete Record Button, with "Are You Sure?" flag	delbtn?ObjectName

Table 4: Example Object Naming Convention

7.3.2 DB Schema

When selecting the data base engine and CASE tool for a project, the compatibility of one with the other should be an explicit selection criterion. However, this may spawn a debate regarding whether a CASE tool should influence the data base to be selected. Ironically, the CASE tool is often selected before the data base engine is selected, since some level of design must be completed in order to specify the data base requirements. This problem can be circumvented by using a CASE tool whose schema generator supports a wide variety of data base products and technologies. Chances are, there will be a match between the "best" data base engine based on the design requirements and those which are compatible with the CASE tool. If such good fortune fails to arise, then consideration should be given to switching to a CASE tool that does support the data base of choice, since the project will still be early in its life cycle and the cost of switching CASE tools may be less than that of finishing the project with mismatched tools. (Of course, if the data base has been cast before the project begins, then select a CASE tool that provides a schema generator for it.) For example, Visual Basic now supports Microsoft Access 2.0 and the System Architect CASE tool can generate a "vanilla" SQL that can be used with minor editing to automatically create the data base structures.

Incremental development (a.k.a. Spiral Model and Software Accretion) is becoming a common strategy, especially when using modern tools and languages such as those under investigation. One challenge of this strategy is frequent design changes based on "lessons learned" from the prior increment. This, in turn, creates difficulties in keeping the design synchronized with the current version of the software. For synchronizing the design representation with the "as-built" application during incremental development, several approaches are suggested:

1. Settle on design conformity/leniency rules (i.e., how far can the programmer deviate from the design without invoking a redesign cycle) and design update

frequencies for manually synchronizing the design (this approach applies whether or not a CASE tool is being used).

2. This problem may be mitigated by deciding to carry a minimum of detail in the design, leaving a great deal of leeway for the programmer during implementation. However, this approach adds a significant design and documentation burden on the programmer. The programming staff must be good at designing code modules and be religious in the documentation of their as-built code.

3. Select the CASE tool and data base such that reverse engineering can be used to convey changes implemented in the development tool back to the CASE tool.

Know your tool's presumptions about how an application will be developed and go with the flow. By simply understanding the expected sequence of development, one can streamline the development process. Conversely, bucking the system can easily cripple an otherwise useful tool. This is not to say that one should use risky or unsatisfactory development practices. And, of course, never, never, never would I suggest that one change the problem to suit the tool. However, if one approach is about the same as another, then let the expected synergy with the development tool make the decision. (This, in turn, means that someone on the development team must know, or be able to find out, how the tool expects the problem to be tackled.) For example, ObjectVision assumes the sequence of development will be: 1) Layout the User Interface, 2) Program the Operational Behaviors, and 3) Create/Connect the Data Base(s). When this pattern is followed, the data base back-end practically "writes itself" since ObjectVision drafts a "straw man" version of the data base schema based on the existing GUI objects. It even suggests data types and length based on how its associated GUI object has been laid out. Thus, the developer must merely remove or edit the schema elements. If one begins by laying out the data base, every element will have to be put in by hand. This example should be contrasted with PowerBuilder, which presumes that a data base already exists and attempts to aid the developer in building the front end and which can't do much more than sketch out user screens unless an underlying data base is actually available.

7.3.3 Event-Based and/or Object-Oriented Design

All of the visual programming tools and languages investigated for this effort employed event-based processing and object-oriented programming at least to some degree. Some just scratch the surface of these non-traditional programming paradigms (e.g., Visual Basic) while others are quite mature (e.g., Smart Elements). Thus, it is necessary to discuss potential conflicts and synergies which arise not because of the visual nature of the tool or language, but due to the intrinsic use of these emerging programming paradigms. Further, if the implementation tool/language is in fact known at design-time, avoid fighting the language; it is better to adapt (limit) the design methodology to take advantage of whatever advanced features are available (e.g., object-orientation, or event-based processing) and use procedural or conventional approaches for the balance. Two examples of such prudence follow:

1. When Visual Basic is known to be the implementation language, the design should be geared to have only one property per object causing event-based behaviors to execute, since the "Tag" property does not spawn events and no other value-properties can be added. (This example presumes that the developer is <u>not</u> a C programmer capable of constructing a custom control.)

2. When Visual Basic is known to be the implementation language, one should avoid the use of objects other than those destined for the User Interface, since

> Visual Basic does not support OOP in the general sense. Another approach (not tested) is to create a Virtual Form to hold objects which will be used internally but never actually displayed to the User; this would in essence "trick" Visual Basic into having a collection of objects for use (literally) "behind the scenes."

If you are going to use an object-oriented tool to implement an application, Go For The Gold in the design process. Craft Methods that are as generic as possible. Apply them as high in the hierarchy as possible. Take full advantage of classification structures and let the benefits of OOP shine through.

For fully equipped OOP environments, it may be preferable to keep (reasonably related) communication links between major system modules simple (like a "pinch point"). A single, simple message from one module to the other is easy to follow and debug if problems do arise. The target method can then spawn however complex a set of processes as are required (see Figure 9). This advice may also be useful in designing communications between modules in a distributed application using a client/server architecture.

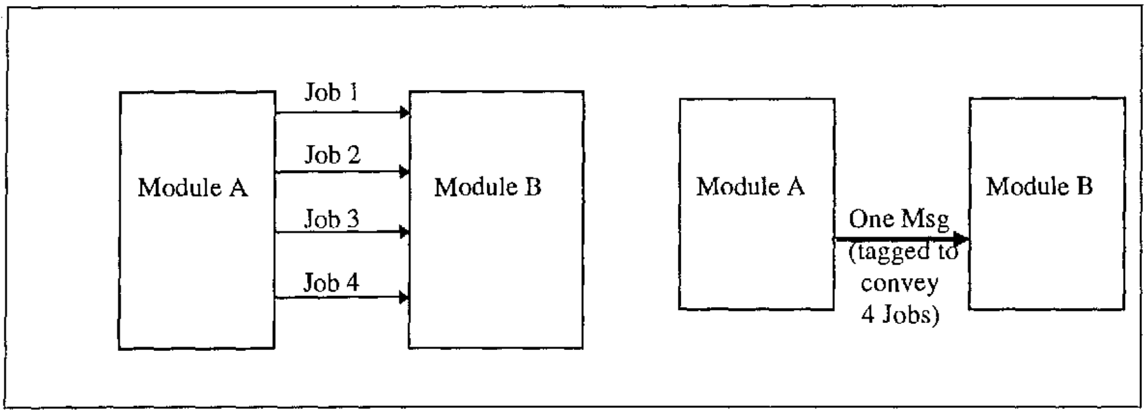


Figure 9: Individual versus Coalesced Messaging

Smart Elements does not provide a vehicle to explicitly notify user interface objects when they to be updated. Note that for objects having a one-to-one correlation between the knowledge base and user interface, it does provide a linking mechanism; however, it is often the case that one would want to send a message from one Knowledge Base object to a non-correlated GUI object. This type of messaging is provided in the other direction. In the Tic-Tac-Toe application, the Knowledge Base and GUI objects could not be linked because the object value in the GUI was symbolic (X, O or <space>), whereas in the Knowledge Base, the value was numeric (+1,-1 or 0).] The workaround used in the Tic-Tac-Toe application causes a great deal of extra work, since every interface object must be "pulsed" after each call to the Engine to see if they need to "do anything." Other tools suffer from similar front-end/back-end communication gaps, such as opportunistically advising a User Interface when a stored procedure has placed new data into the data base. In a full strength application, where performance could be in jeopardy, a more focused (i.e., intelligent rather than exhaustive) messaging system would have to be crafted. For example, one could add a "black board" table into which the data base "engine" could place a list of updated data objects and which an Interface method could use to update just those Interface objects whose data values had changed. Of course in the Smart Elements application, the built-in linking mechanism could have been used "as is" and then have an Interface method convert the numeric value into its symbolic equivalent.

7.3.4 Function Design

Consider the use of a tool that provides an explicit rule-based paradigm, even if the application is not an expert system or does not require inferencing. The rules can be used to expedite control strategy/logic or to explicitly represent the business rules to be followed. The visualization of such rules can be a powerful communication tool for use with the internal customer how the program will behave. Visualization of processing logic can also accelerate the validation of the program by a testing or design review group.

To accommodate nuances and/or unknowns of the implementation tool, the design must be kept generic (tool-free) down to a point. Then, if the tool and its special needs are known, a layer of specialization can be added. (Note that the Gane & Sarson process modeling technique uses a drill-down approach to specificity, thus making it suitable for this approach.)

For applications which include an underlying data base (probably relational), consider using a hybrid of software modeling methodologies. In particular, Gane & Sarson Data Flow Diagrams for high-level context and major processes, Entity-Relationship Diagrams for detailed data schema, and Coad/Yourdon Event Diagrams for events make a good combination.

7.4 Summary of Findings

Table 5 is a matrix which presents the observation results (from Section 7.1 and appendices H and I) mapped into the functional categories developed in Section 7.3. Thus, the table presents the observation frequency of conflicts and synergies as a function of Application and Category. The table indicates that, in general, the current state of the technology provides more instances of conflict than of synergy. Also note that the most advanced tool, Smart

	Categories							
Application Definition	User Interface		DB Schema		Event-Based/ OO Design		Function Design	
	Cnflct	Syn	Cnflct	Syn	Cnflct	Syn	Cnflct	Syn
Customer Support Tracking System (Visual Basic with CASE-based DFDs and ERDs)	2		12				2	
Customer Support Tracking System (ObjectVision with CASE-based DFDs and ERDs)	2		6	3			2	
Tic Tac Toe (Visual Basic with OOA/D)					2	1		
Tic Tac Toe (Smart Elements with OOA/D)	2				2	5		1
Peer Observations (with CASE)	often		often	often		often	usual	
Peer Observations (CASE not relevant)							often	

Table 5: Frequency of Conflicts and Synergies

Elements, provided more instances of synergy than conflict. Perhaps this is an indication that these technologies are indeed beginning to mature.

Applying a software engineering methodology provided benefits during the design and implementation of the test bed applications. The design process surfaced data structure and behavioral issues that would have not otherwise have been discovered until the debugging stage had begun. As such, there seemed to be considerably fewer hours spent writing and debugging code compared to other programming projects undertaken by this author. When implementation problems did occur, reference back to the design documents usually helped solve them. Although not formally studied, the author also believes that the conclusions of the research would have been the same no matter which of the mainstream software engineering methodologies had been chosen. Thus, the decision to use a software engineering methodology is more critical than the choice of which one or whether to use a CASE tool to implement it.

Using a visual programming language or tool provided benefits during the development of the test bed applications. It would seem that this would always be the case if the application has a visual component (e.g., a GUI) or is such that visualization of its design and/or operation is important (e.g., model-based reasoning or simulation). Productivity was higher when using the tools than when using Visual Basic. However, not all tools offer the same flexibility. For example, for ObjectVision to deliver a net benefit, the application must closely fit the expected mold. Conversely, Smart Elements can be made to look and feel more like a language than a tool when the built-in functions and features are not sufficient.

Ironically, even though it was manipulating GUI resource files, the SA2VB bridge itself had no visual dimension to it. Thus, it was implemented in a non-visual development environment. Further, since it required no complex data structures and no complex architecture, hand-marked example input and output files and hand-sketched logic diagrams were the extent of analysis and design required to solve the problem. The lesson here is that, as powerful as CASE tools and visual development environments may be, there are still cases where the complexity of the problem does not warrant the investment required to procure and learn how to use them.

Interpolating between the test bed applications, which clearly benefited from both the application of software engineering methods and visual development tools, and bridge, which did not, leads to the possibility that there lies a class of problems which can and should be solved using the visual development environment alone. An example of such might be the bridge application with the added requirements of a GUI-based file browser and preview capability. Conversely, a "pure" data base application (perhaps CSTS without the call timer and with a simplified user interface) could be designed in a CASE tool and generated by it with little or no additional programming. However, the relative number of problems whose solution fit one of these profiles may be small, such that the best advice is to establish a development environment that provides a flexible, cooperative suite of software engineering methodologies and visual programming languages and tools. From there, standards can be developed as to which tools and methodologies in the suite should be applied to which problems.

Guidelines and utilities fashioned along the lines of those presented in this thesis should be directly beneficial to developers charged with delivering an application using a visual language or tool while following a formal software engineering methodology. This will be especially true for projects involving a team of developers. Of more importance, such guidelines and utilities are themselves primary ingredients of the merged CASE and visual programming environments of the future. Perhaps the results presented here will facilitate the transition.

REFERENCES

[Blum94]
Blum, Bruce I., "A Taxonomy of Software Development Methods," Communications of the ACM, 37, 11 (November 1994), pp. 82-94.

[Booch91]
Booch, Grady, Object Oriented Design with Applications, The Benjamin/Cummings Publishing Company Inc., 1991.

[Borland91A]
Borland ObjectVision version 2.1, Getting Started, Borland International, Inc., 1991.

[Borland91B]
Borland Turbo C++ version 1.0.1, User's Guide, Borland International, Inc., 1991.

[Braithwaite90]
Braithwaite, Kenmore S., Applications Development Using CASE Tools, Academic Press, Inc., San Diego, 1990.

[Chang90]
Chang, Shi-Kuo, Visual Languages and Visual Programming, Plenum Press, New York, 1990.

[Chang94]
Chang, Carl K., "Changing Face of Software Engineering," IEEE Software, 11, 1 (January 1994), pp. 4-5.

[Coad/Yourdon90]
Coad, Peter and Edward Yourdon, Object-Oriented Analysis, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[Coad/Yourdon91]
Coad, Peter and Edward Yourdon, Object-Oriented Design, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[Constantine94]
Constantine, Larry, "Modeling Matters," Software Development, February, 1994, pp. 96-94.

[Constantine95]
Constantine, Larry, "Shapes to Come," Software Development, May, 1995, pp. 96-95.

[Hanna94]
Hanna, Mary, "New Breed of 4GL Puts Pretty Face on C/S Apps," Software Magazine, 14, 12 (December 1994), pp. 33-40.

[Harding95]
Harding, Elizabeth U., "Will IBM Make Smalltalk?," Software Magazine, 15, 2 (February 1995), pp. 21-22.

[Ichikawa90]
Ichikawa, Tadao, et al, Visual Languages and Applications, Plenum Press, New York, 1990.

[IEEE95]
Burnett, Margaret M., and David W. McIntyre, Guest Editors for Feature Articles on Visual Programming, IEEE Computer, 28, 3, (March 1995), pp. 14-66.

[Jicha94]
Jicha, Henry, "Object Technology Explodes with Visual Programming," Object Magazine, 4, 4 (July-August 1994), pp. 33-36.

[Keuffel94]
Keuffel, Warren, "MicroCASE: A Grass-Roots Strategy for Deploying CASE Tools," Software Development, February, 1994, pp. 37-42.

[Kiyooka95]
Kiyooka, Gen, "Ode To AppWizard," Software Development, January, 1995, pp. 79-81.

[Layout92]
Layout version 3.03, Layout For Programmers, Objects, Inc. 1992.

[Linthicum94]
Linthicum, David S., "Get the Picture with Visual Programming," Application Development Trends, February, 1994, pp. 52-58.

[Lowry92]
Lowry, Michael R., "Software Engineering in the Twenty-First Century," AI Magazine, 14, 3 (Fall 1992), pp. 71-87.

[Martin89]
Martin, James, Information Engineering, Book I: Introduction, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[McConnell93]
McConnell, Steven C., Code Complete: A Practical Handbook of Software Construction, Microsoft Press, Redmond, Washington, 1993.

[Microsoft92]
Microsoft Windows Software Development Kit, Programmer's Reference, Volume 4: Resources, Chapter 7, "Resource Formats Within Executable Files," and Chapter 13, "Resource-Definition Statements," Microsoft Corporation, 1992.

[Microsoft93]
Microsoft Visual Basic Programming System for Windows Version 3.0, Programmer's Guide, Microsoft Corporation, 1993.

[NeuronData94A]
Nexpert Object version 3.0, Nexpert Object: User's Guide, Neuron Data, Inc., 1994.

[NeuronData94B]
Open Interface version 3.0, Open Interface: User's Guide, Neuron Data, Inc., 1994.

[NeuronData94C]
Smart Elements version 2.0 Introduction Manual, Neuron Data, Inc., 1994.

[NeuronData94D]
Nexpert Object version 3.0, Functional Description, Neuron Data, Inc., 1994, pp. 71-80.

[O'Brien94]
O'Brien, Larry, "CASE's Gordian Knot," Software Development, February, 1994, pp. 7-10.

[Petzoid92]
Petzoid, Charles, Programming Windows 3.1 PART 3 Using Resources, Chapter 10 "Dialog Boxes," <???>, <???>, 1992.

[Pressman92]
Pressman, Roger S., Software Engineering: a Practitioner's Approach, Third Edition, McGraw-Hill, Inc., New York, 1992.

[Rich93]
Rich, Charles, and R. C. Waters, "Approaches to Automatic Programming," Advances in Computers, Marshall C. Yovits, ed., Academic Press, Inc., San Diego, 1993.

[Schmidt95]
Schmidt, Jennifer, "Choice of Visual Tool Depends on Focus," Application Development Trends, April, 1995, pp. 31-40.

[Shu88]
Shu, Nan C., Visual Programming, Van Nostrand Reinhold Company, New York, 1988.

[SysArch94A]
System Architect User Guide & Reference Manual, Popkin Software & Systems Incorporated, 1994.

[SysArch94B]
System Architect Screen Painter, Section 5.4 "Generating .DLG and .H Files," Popkin Software & Systems Incorporated, 1994.

[West92]
West, M., "How Object Oriented Is Your Visual Programming Tool?," GartnerGroup Applications Development & Management Strategies Research Note, T-700-794, November 23, 1992.

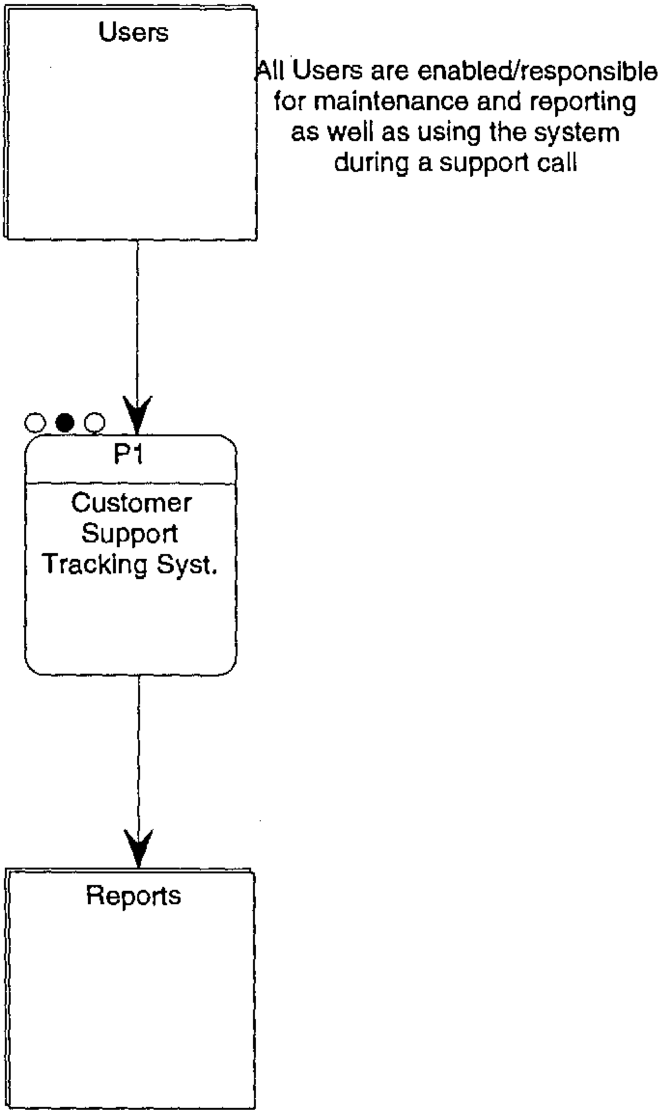
APPENDIX A

Customer Support Tracking System Design Package

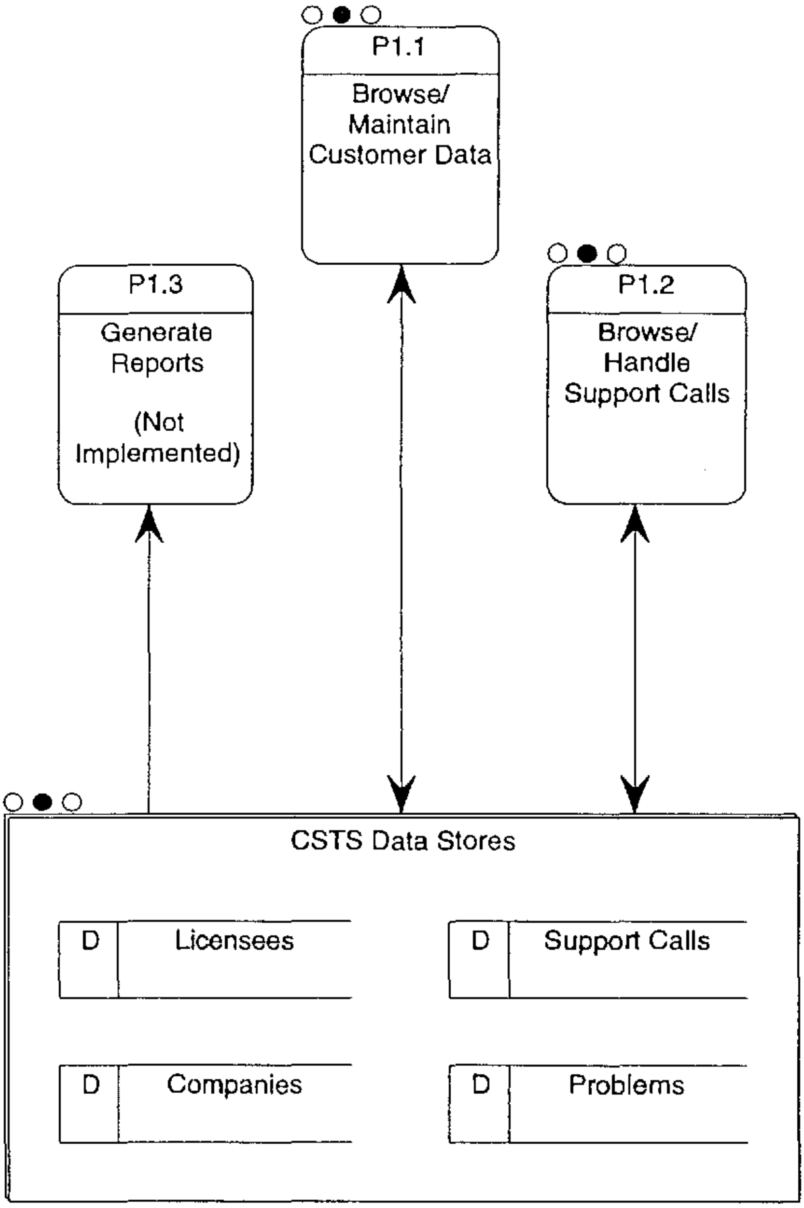
CSTS Requirements Definition Statement

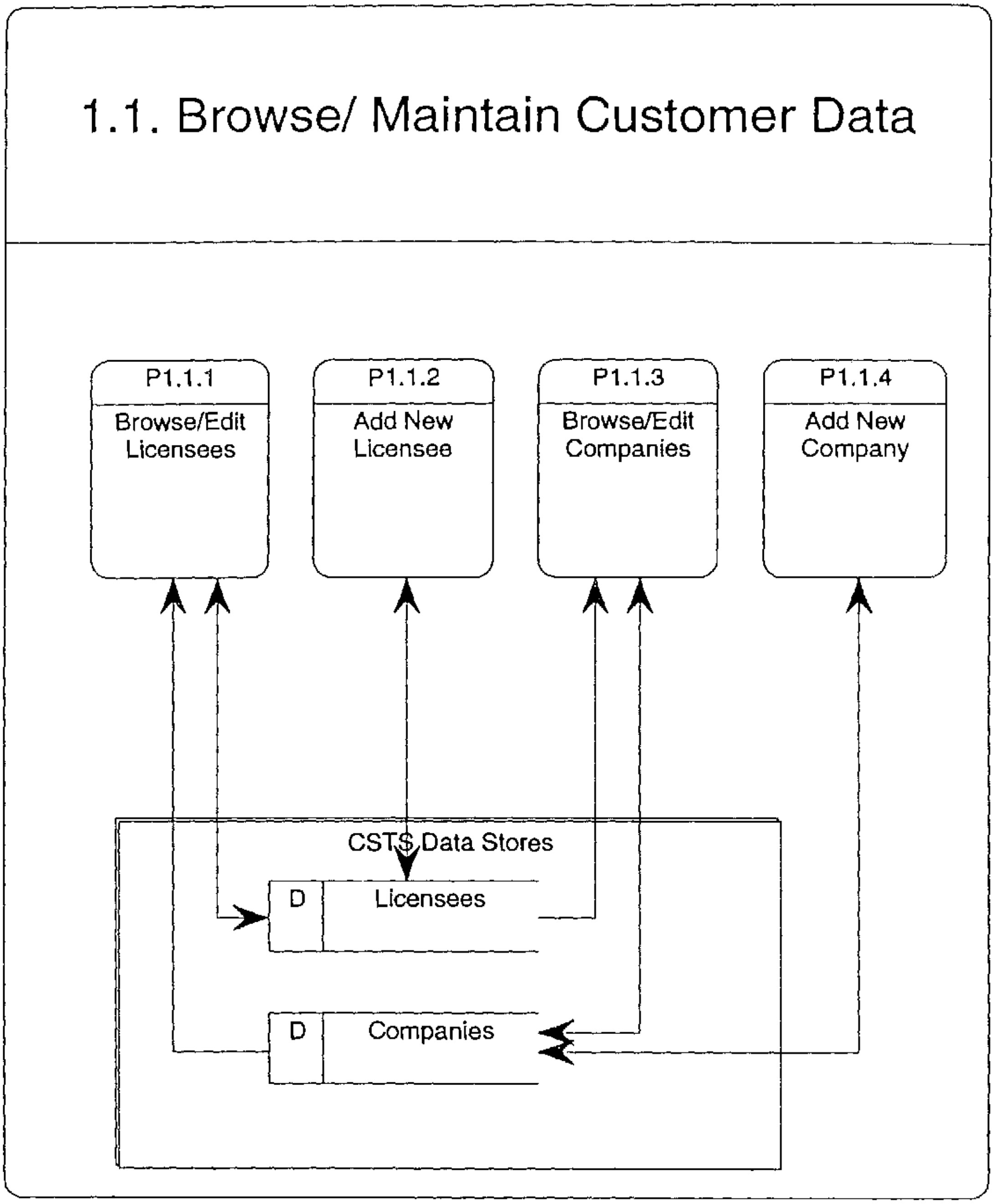
Upon startup, the system shall present to the user a form-like data entry screen, plus several options available from either menus or buttons. The main data entry screen shall be named "Customer Support Tracking System" and shall provide a place for a User ID (which the system must guarantee as unique), Company Name, Address (two lines, plus City, State, and Zip+4), Country, Telephone (with 5 digit extension) and FAX, Contact Name and Title, Date First Product Shipped and the Total Support Time rendered. Information entered using this screen shall be stored in a CUSTOMER data base using User ID as the primary key. The system shall be designed such that a Customer's primary record may be both created and maintained using this same screen. The Total Support Time area shall not be user editable, but rather shall be calculated by the system each time support is provided; the system shall provide an "Update Total Support Time" menu option under a "Maintenance" menu bar item in case it must be overridden by the user. The other functions under "Maintenance" shall be "Delete Company" and "Delete Licensee." The Main screen shall provide a <Support Calls> button which shall take the user to the "Customer Support" screen. The Customer Support screen shall also appear form-like and shall repeat the Licensee ID and Contact Name from the customer's primary record. It shall automatically provide the Support Date and Time for the support currently being provided, plus a scrollable, unlimited, editable text field for capturing Comments, the Elapsed Time. and a user definable Combo list of Support Types. The Customer Support screen shall provide buttons for starting and stopping a timer and for returning to the Main screen. The system shall maintain a SUPPORT data base containing the data from individual support entries with the Date/Time stamp as the unique Primary Key and the User ID as the Foreign Key (to CUSTOMER). Returning to the Main screen shall also cause the system to increment the Total Support Time field by the amount of time in the Elapsed field.

Context
Customer Support Tracking System

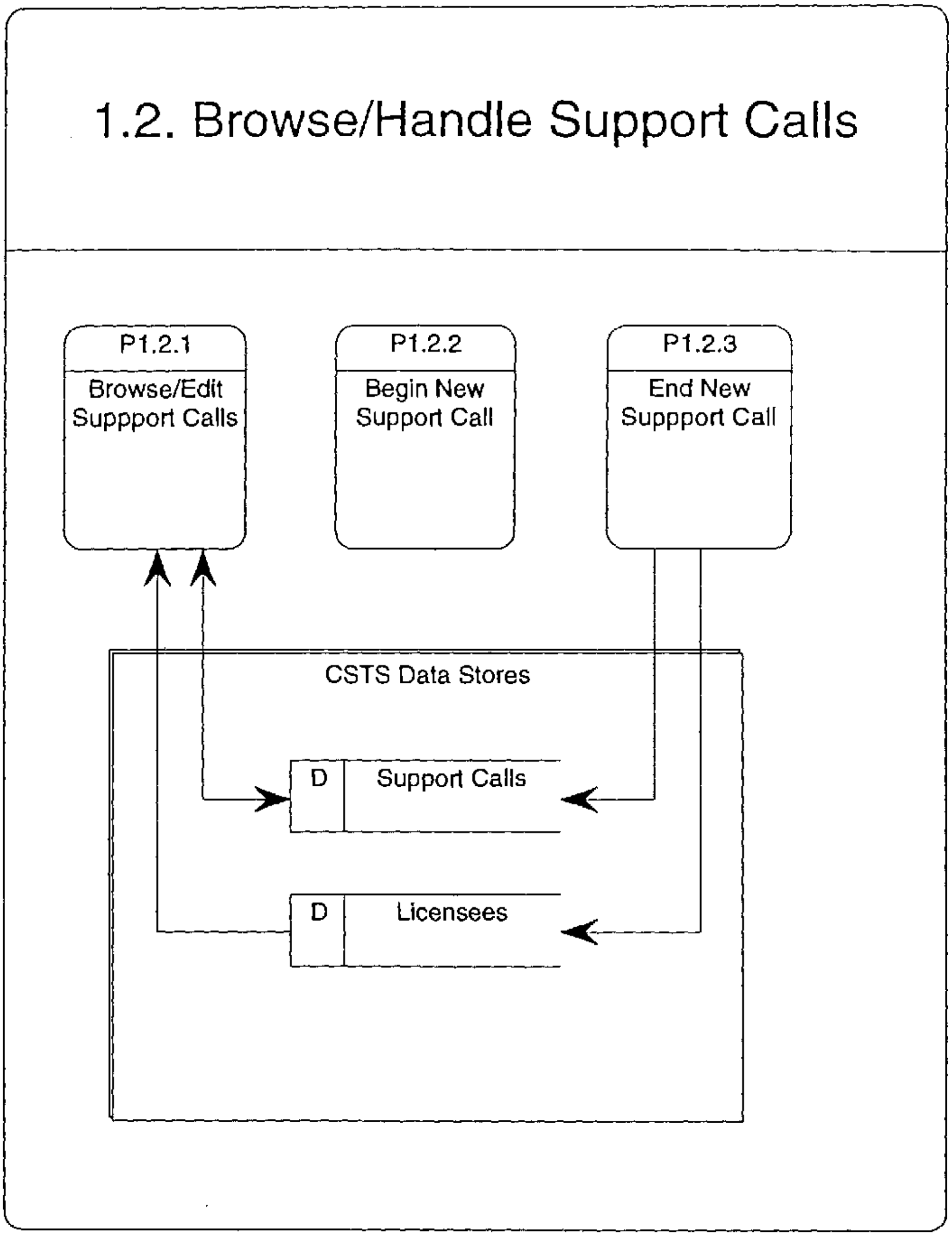

Users
All Users are enabled/responsible for maintenance and reporting as well as using the system during a support call
○●○
P1
Customer Support Tracking Syst.
Reports

Major Processes
Customer Support Tracking System





1.1. Browse/ Maintain Customer Data


1.2. Browse/Handle Support Calls
P1.2.1
Browse/Edit Suppport Calls
P1.2.2
Begin New Support Call
P1.2.3
End New Suppport Call
CSTS Data Stores
D
Support Calls
D
Licensees

Name: Add New Company
Purpose:
This process finds the last record in Companies, reads its value for CMPY_LOC_ID, increments by 1, creates a new record with that ID sets the current record pointer to it, and returns control to the Browse/Edit Company Information process.

Documentation:
Responsible:
Transaction Frequency : **StartDt:** **CompDate:**

Description:

Name: Add New Licensee
Purpose:
This process allows the user to enter a new Licensee ID and then ensures that it is unique. If so, this process creates a new (empty) licensee record with that ID. If not, it sets the current record pointer to that licensee. Control is then returned to Browse/Edit Licensees.

Documentation:
Responsible:
Transaction Frequency : **StartDt:** **CompDate:**

Description:

Name: Begin New Support Call
Purpose:
Save current time stamp. Start timer object if appropriate.

Documentation:
Responsible:
Transaction Frequency : **StartDt:** **CompDate:**

Description:

Name: Browse/Edit Companies
Purpose:
The Licensee and Company tables shall be joined via Company/Location ID(CMPY_LOC_ID). Company data may be edited directly. The user shall have the ability to page up and down thru the records, go to the top or the bottom of the records. The user may invoke commands to "Accept" (store changes), "Cancel", or "Add New Company".

Documentation:
Responsible:
Transaction Frequency : **StartDt:** **CompDate:**

Description:

Name: Browse/Edit Licensees
Purpose:
The Licensee and Company tables shall be joined via Company/Location ID(CMPY_LOC_ID). Licensee data may be edited directly; editing of Company data shall require the user to invoke a "Change Company Info" command. The user shall have the ability to page up and down

thru the records, go to the top or the bottom of the records, delete a Licensee, or delete a Company. The user my invoke commands for "Add New Licensee", "Add New Company" or "Support Call".

Documentation:
Responsible:
Transaction Frequency : **StartDt:** **CompDate:**

Description:

Name: Browse/Edit Suppport Calls
Purpose:
Filter records based on current LICENSEE_ID in Licensees. Support data may be edited directly. The user shall have the ability to page up and down thru the records, go to the top or the bottom of the records, or delete a record. The user my invoke commands for "New Support Call", or "Done".

Documentation:
Responsible:
Transaction Frequency : **StartDt:** **CompDate:**

Description:

Name: End New Suppport Call
Purpose:
Clock Duration of Support Call and Store its value (in minutes) in SPRT_TM of the current (new) support record. Calculate a new summation of the support time for the current Licensee and store it in LIC_TOT_SPRT_TM of that licensee's master record. Close the form and return control to Browse/Edit Customers.

Documentation:
Responsible:
Transaction Frequency : **StartDt:** **CompDate:**

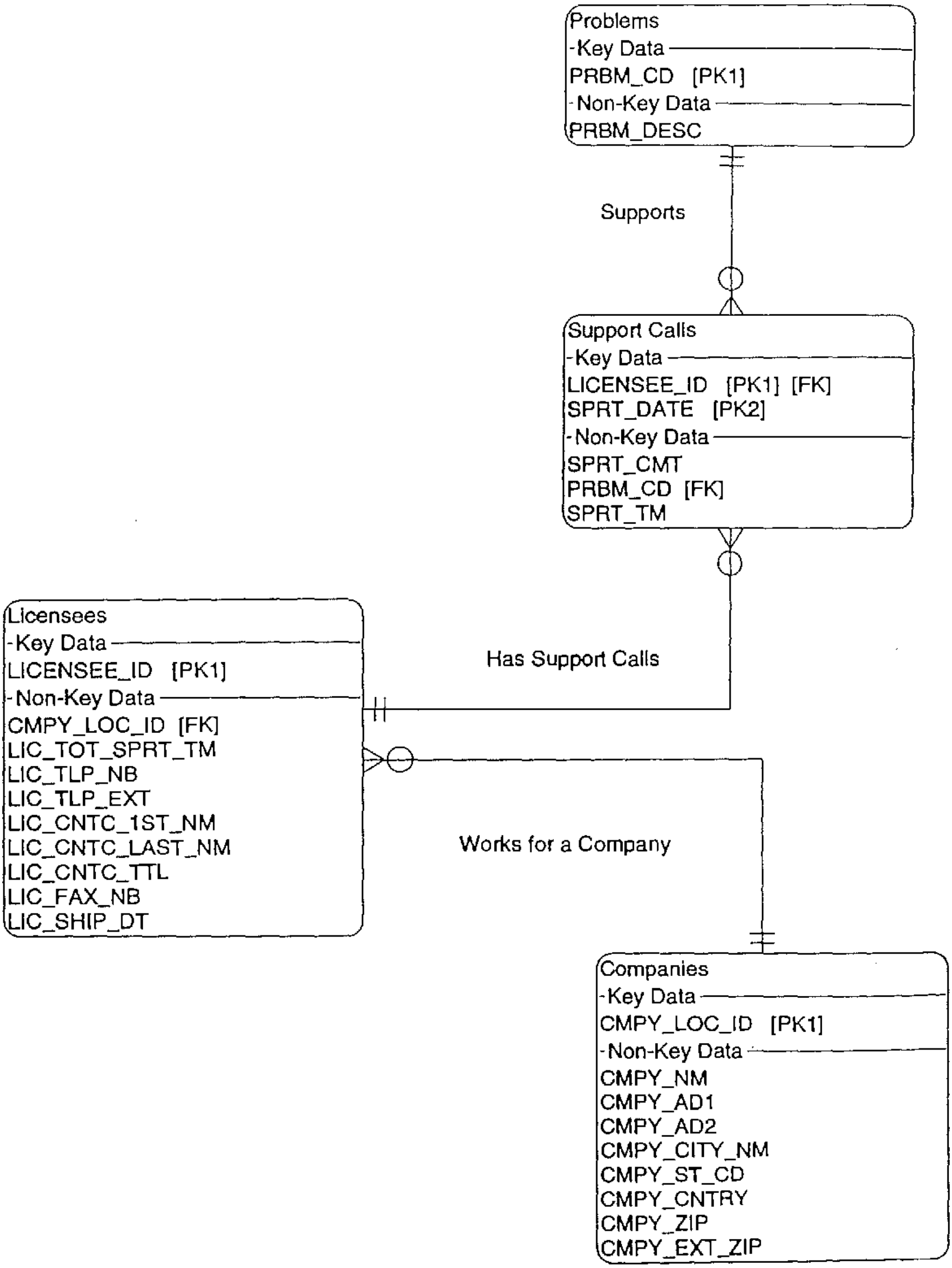
Description:

Name: Maintain Customer Data
Purpose:

Documentation:
Responsible:
Transaction Frequency : **StartDt:** **CompDate:**

Description:

CSTS Data Stores



CSTS Data Stores

```
Companies                                         Entity
  Volume:                                         Normalize: T
  Comments:

  Purpose:

    CMPY_AD1
      Type: CHARACTER                       Width: 40
      Domain:                               Length:
      Description:
      Customer Address Part One

      Comments:

    CMPY_AD2
      Type: CHARACTER                       Width: 40
      Domain:                               Length:
      Description:
      Customer Address Line Two

      Comments:

    CMPY_CITY_NM
      Type: CHARACTER                       Width: 25
      Domain:                               Length:
      Description:
      Customer City Name

      Comments:

    CMPY_CNTRY
      Type: CHARACTER                       Width: 15
      Domain:                               Length:
      Description:
      Customer Country

      Comments:

    CMPY_EXT_ZIP
      Type: INTEGER                         Width:
      Domain:                               Length:
      Description:
      Plus 4 Zip Extension

      Comments:

    CMPY_LOC_ID
      Type: INTEGER                         Width: 4
      Domain:                               Length:
      Description:
      Unique Company/Location ID; automatically incremented as new record is added;
      supports multiple users at same company and location, using the same license ID.

      Comments:

    CMPY_NM
      Type: CHARACTER                       Width: 50
      Domain:                               Length:
```

CSTS Data Stores

Description:
Company name.

Comments:

CMPY_ST_CD

```
Type: CHARACTER                    Width: 2
Domain:                            Length:
Description:
Customer State Code

Comments:
```

CMPY_ZIP

```
Type: CHAR                         Width: 5
Domain:                            Length:
Description:
Customer Zip Number

Comments:
```

CSTS Data Stores

```
Licensees                                            Entity
  Volume:                                            Normalize: T
  Comments:

  Purpose:

    CMPY_LOC_ID
      Type: INTEGER                        Width: 4
      Domain:                              Length:
      Description:
      Unique Company/Location ID; automatically incremented as new record is added;
      supports multiple users at same company and location, using the same license ID.

      Comments:

    LICENSEE_ID
      Type: CHARACTER                      Width: 28
      Domain:                              Length:    4
      Description:
      Company's ID which will be assigned when the Software package is shipped.  This
      number can be found in the runtime about box, for applications which support
      embedded User IDs.

      Comments:

    LIC_CNTC_1ST_NM
      Type: CHARACTER                      Width: 20
      Domain:                              Length:
      Description:
      Customer contact first name.

      Comments:

    LIC_CNTC_LAST_NM
      Type: CHARACTER                      Width: 20
      Domain:                              Length:
      Description:
      Customer contact last name.

      Comments:

    LIC_CNTC_TTL
      Type: CHARACTER                      Width: 40
      Domain:                              Length:
      Description:
      Customer Contact Title Name

      Comments:

    LIC_FAX_NB
      Type: CHARACTER                      Width: 14
      Domain:                              Length:
      Description:
      Customer FAX Telephone Number

      Comments:

    LIC_SHIP_DT
  Type: DATE                              Width:
  Domain:                                 Length:
  Description:
  Shipping date for the original runtime package.

  Comments:
```

LIC_TLP_EXT

```
  Type: CHARACTER                         Width: 5
  Domain:                                 Length:
  Description:
  Customer Telephone Extension Number

  Comments:
```

LIC_TLP_NB

```
  Type: CHARACTER                         Width: 14
  Domain:                                 Length:
  Description:
  Customer Telephone Number

  Comments:
```

LIC_TOT_SPRT_TM

```
  Type: INT                               Width:
  Domain:                                 Length:
  Description:
  Total support time for this customer in minutes.

  Comments:
```

CSTS Data Stores

```
Problems                                             Entity
  Volume:                                            Normalize: T
  Comments:

  Purpose:

    PRBM_CD
      Type: CHARACTER                         Width: 8
      Domain:                                 Length:
      Description:
      Problem Code will contain the code for the recurring instances of support given, for
      example:  problems with installation, or configuration.

      Comments:

    PRBM_DESC
      Type: TEXT                              Width:
      Domain:                                 Length:
      Description:
      Description of a recurring problem.

      Comments:
```

CSTS Data Stores

Support Calls Entity
Volume: Normalize: T
Comments:

Purpose:

LICENSEE_ID
Type: CHARACTER Width: 28
Domain: Length: 4
Description:
Company's ID which will be assigned when the Software package is shipped. This number can be found in the runtime about box, for applications which support embedded User IDs.

Comments:

PRBM_CD
Type: CHARACTER Width: 8
Domain: Length:
Description:
Problem Code will contain the code for the recurring instances of support given, for example: problems with installation, or configuration.

Comments:

SPRT_CMT
Type: TEXT Width:
Domain: Length:
Description:
This is a memo field to contain the comment/reason for the support.

Comments:

SPRT_DATE
Type: DATE Width:
Domain: Length:
Description:
Contain the date of the support.

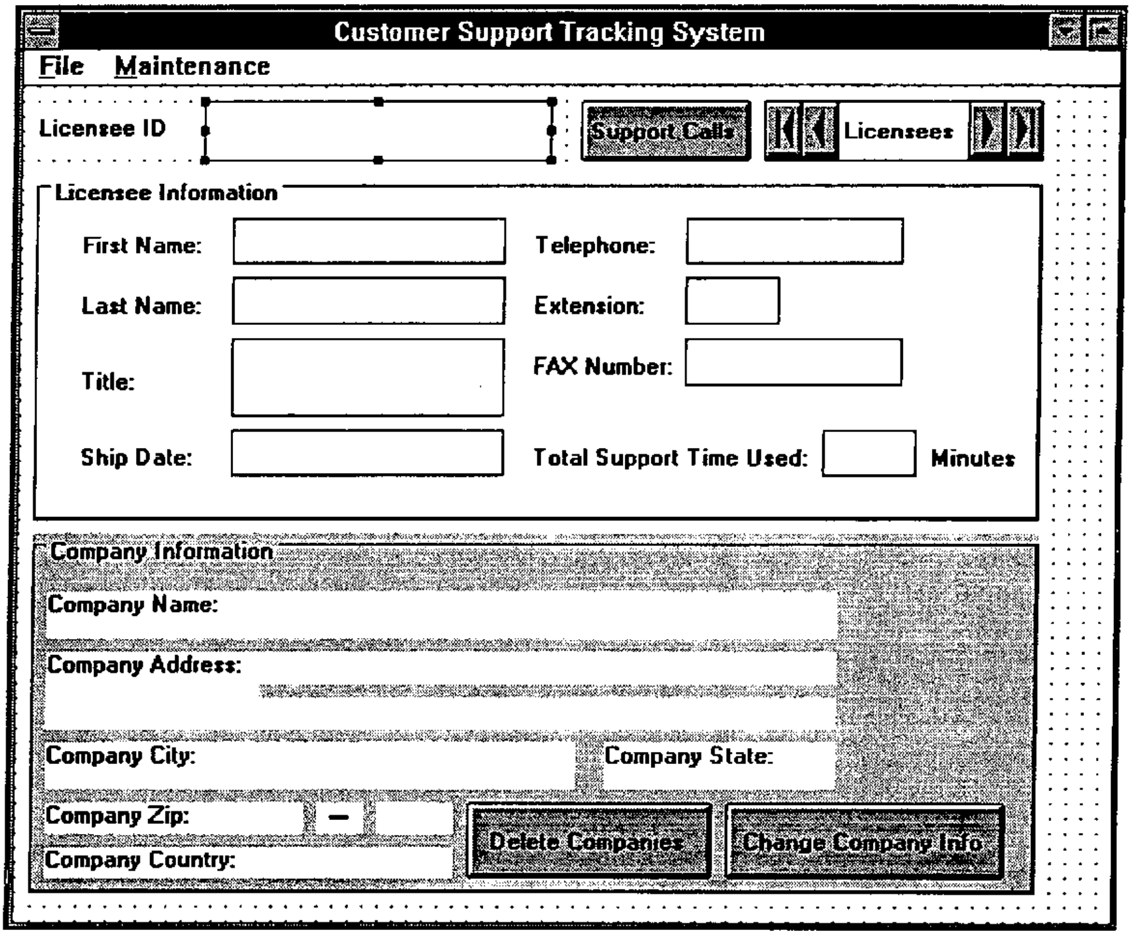
Comments:

SPRT_TM
Type: INT Width:
Domain: Length:
Description:
Support Time will contain the duration of support time in minutes.

Comments:

APPENDIX B

Customer Support Tracking System Visual Basic Listings/Screens



Customer Support Tracking System Main Screen (Visual Basic Version)

Main Form Object Definitions (CSTS.FRM)

```
VERSION 2.00
Begin Form CSTSMain
  Caption       =  "Customer Support Tracking System"
  ClientHeight   =  6735
  ClientLeft     =  360
  ClientTop      =  1605
  ClientWidth    =  8640
  Height         =  7425
  Left          =  300
  LinkTopic      =  "Form1"
  ScaleHeight    =  6735
  ScaleWidth     =  8640
  Top           =  975
  Width          =  8760
  Begin CommandButton btnSprtCall
    Caption        =  "Support Calls"
    Height         =  495
    Left          =  4440
    TabIndex        =  40
    Top            =  120
    Width          =  1335
  End
  Begin Frame CMPYData
    BackColor      =  &H00E0E0E0&
    Caption        =  "Company Information"
    Height         =  2895
    Left          =  120
    TabIndex        =  18
    Top            =  3600
    Width          =  7935
    Begin CommandButton btnCoMaint
      Caption        =  "Delete Companies"
      Height         =  615
      Left          =  3480
      TabIndex        =  9
      Top            =  2160
      Width          =  1935
    End
    Begin CommandButton btnChgCoInfo
      Caption        =  "Change Company Info"
      Height         =  615
      Left          =  5520
      TabIndex        =  10
      Top            =  2160
      Width          =  2175
    End
    Begin Label Label2
      Caption        =  "Company Country:"
      Height         =  255
      Left          =  120
      TabIndex        =   38
      Top             =   2520
      Width           =   1575
   End
   Begin Label Label1
      DataField       =   "CMPY_CNTRY"
      DataSource      =   "Licensees"
      Height          =   255
      Left            =   1680
      TabIndex        =   39
      Top             =   2520
      Width           =   1695
   End
   Begin Label CompanyName
      DataField       =   "CMPY_NM"
      DataSource      =   "Licensees"
      Height          =   375
      Left            =   1560
      TabIndex        =   22
      Top             =   480
      Width           =   4815
   End
   Begin Label CompanyAddr1
      DataField       =   "CMPY_AD1"
      DataSource      =   "Licensees"
      Height          =   255
      Left            =   1800
      TabIndex        =   23
      Top             =   960
      Width           =   4575
   End
   Begin Label CompanyAddr2
      DataField       =   "CMPY_AD2"
      DataSource      =   "Licensees"
      Height          =   255
      Left            =   1800
      TabIndex        =   13
      Top             =   1320
      Width           =   4575
   End
   Begin Label CompanySt
      DataField       =   "CMPY_ST_CD"
      DataSource      =   "Licensees"
      Height          =   375
      Left            =   6000
      TabIndex        =   14
      Top             =   1680
      Width           =   375
   End
   Begin Label CompanyCity
      DataField       =   "CMPY_CITY_NM"
      DataSource      =   "Licensees"
      Height          =   375
      Left          =   1440
      TabIndex      =   37
      Top           =   1680
      Width         =   2895
   End
   Begin Label CmpyZPEXT
      DataField     =   "CMPY_EXT_ZIP"
      DataSource    =   "Licensees"
      Height        =   255
      Left          =   2760
      TabIndex      =   36
      Top           =   2160
      Width         =   615
   End
   Begin Label lblCmpySt
      Caption       =   "Company State:"
      Height        =   375
      Left          =   4560
      TabIndex      =   35
      Top           =   1680
      Width         =   1455
   End
   Begin Label lblCmpyCity
      Caption       =   "Company City:"
      Height        =   375
      Left          =   120
      TabIndex      =   34
      Top           =   1680
      Width         =   1335
   End
   Begin Label lblCmpyAd1
      Caption       =   "Company Address:"
      Height        =   615
      Left          =   120
      TabIndex      =   33
      Top           =   960
      Width         =   1695
   End
   Begin Line Line5
      BorderWidth   =   2
      X1            =   2400
      X2            =   2520
      Y1            =   2280
      Y2            =   2280
   End
   Begin Label CompanyZip
      Alignment     =   1  'Right Justify
      DataField     =   "CMPY_ZP"
      DataSource    =   "Licensees"
      Height        =   255
      Left          =   1320
      TabIndex      =   21
      Top           =   2160
```

```
    Width         =  855
  End
  Begin Line Line4
    BorderWidth   =  3
    X1          =  7920
    X2          =  7920
    Y1          =  2880
    Y2          =  120
  End
  Begin Line Line3
    BorderWidth   =  3
    X1          =  0
    X2          =  7920
    Y1          =  2880
    Y2          =  2880
  End
  Begin Label lblHyph
    Caption       =  " "
    FontBold       =  -1  'True
    FontItalic     =  0  'False
    FontName       =  "MS Sans Serif"
    FontSize       =  12
    FontStrikethru =  0  'False
    FontUnderline  =  0  'False
    Height        =  255
    Left        =  2280
    TabIndex       =  15
    Top          =  2160
    Width         =  375
  End
  Begin Label lblCmpyName
    Caption       =  "Company Name:"
    Height        =  375
    Left        =  120
    TabIndex       =  20
    Top          =  480
    Width         =  1455
  End
  Begin Label lblCmpyZP
    Caption       =  "Company Zip:"
    Height        =  255
    Left        =  120
    TabIndex       =  19
    Top          =  2160
    Width         =  1215
  End
End
Begin Frame LicData
  Caption       =  "Licensee Information"
  Height        =  2775
  Left        =  120
  TabIndex       =  16
  Top          =  720
```

```
Width          =  7935
Begin TextBox LicTST
   DataField      =  "LIC_TOT_SPRT_TM"
   DataSource     =  "Licensees"
   Height         =  375
   Left           =  6240
   TabIndex       =  30
   TabStop        =  0   'False
   Top            =  2040
   Width          =  735
End
Begin TextBox LicFAX
   DataField      =  "LIC_FAX_NB"
   DataSource     =  "Licensees"
   Height         =  375
   Left           =  5160
   TabIndex       =  7
   Top            =  1320
   Width          =  1695
End
Begin TextBox LicTE
   DataField      =  "LIC_TLP_EXT"
   DataSource     =  "Licensees"
   Height         =  375
   Left           =  5160
   TabIndex       =  6
   Top            =  840
   Width          =  735
End
Begin TextBox LicTN
   DataField      =  "LIC_TLP_NB"
   DataSource     =  "Licensees"
   Height         =  375
   Left           =  5160
   TabIndex       =  5
   Top            =  360
   Width          =  1695
End
Begin TextBox LicSD
   DataField      =  "LIC_SHIP_DT"
   DataSource     =  "Licensees"
   Height         =  375
   Left           =  1560
   TabIndex       =  4
   Top            =  2040
   Width          =  2175
End
Begin TextBox LicTtl
   DataField      =  "LIC_CNTC_TTL"
   DataSource     =  "Licensees"
   Height         =  615
   Left           =  1560
   MultiLine      =  -1  'True
```

```
      TabIndex        =   3
      Top             =   1320
      Width           =   2175
   End
   Begin TextBox LicLN
      DataField       =   "LIC_CNTC_LAST_NM"
      DataSource      =   "Licensees"
      Height          =   375
      Left            =   1560
      TabIndex        =   2
      Top             =   840
      Width           =   2175
   End
   Begin TextBox LicFN
      DataField       =   "LIC_CNTC_1ST_NM"
      DataSource      =   "Licensees"
      Height          =   375
      Left            =   1560
      TabIndex        =   1
      Top             =   360
      Width           =   2175
   End
   Begin Label lblLicTSTUnits
      Caption         =   "Minutes"
      Height          =   255
      Left            =   7080
      TabIndex        =   32
      Top             =   2160
      Width           =   735
   End
   Begin Label lblLicTST
      Caption         =   "Total Support Time Used:"
      Height          =   255
      Left            =   3960
      TabIndex        =   31
      Top             =   2160
      Width           =   2295
   End
   Begin Label lblLicFAX
      Caption         =   "FAX Number:"
      Height          =   255
      Left            =   3960
      TabIndex        =   29
      Top             =   1440
      Width           =   1215
   End
   Begin Label lblLicTE
      Caption         =   "Extension:"
      Height          =   255
      Left            =   3960
      TabIndex        =   28
      Top             =   960
      Width           =   1215
```

```
 End
 Begin Label lblLicTN
   Caption        =  "Telephone:"
   Height         =  255
   Left         =  3960
   TabIndex       =  27
   Top          =  480
   Width          =  1215
 End
 Begin Label lblLicSD
   Caption        =  "Ship Date:"
   Height         =  255
   Left         =  360
   TabIndex       =  26
   Top          =  2160
   Width          =  1215
 End
 Begin Label lblLicTtl
   Caption        =  "Title:"
   Height         =  255
   Left         =  360
   TabIndex       =  25
   Top          =  1560
   Width          =  1215
 End
 Begin Label lblLicLN
   Caption        =  "Last Name:"
   Height         =  255
   Left         =  360
   TabIndex       =  24
   Top          =  960
   Width          =  1215
 End
 Begin Line Line2
   BorderWidth    =  3
   X1          =  7920
   X2          =  7920
   Y1          =  2760
   Y2          =  120
 End
 Begin Line Line1
   BorderWidth    =  3
   X1          =  0
   X2          =  7920
   Y1          =  2760
   Y2          =  2760
 End
 Begin Label lblLicFN
   Caption        =  "First Name:"
   Height         =  255
   Left         =  360
   TabIndex       =  17
   Top          =  480
```

```
      Width       =  1215
    End
  End
  Begin TextBox CompanyIDFK
    DataField     =  "Licensees.CMPY_LOC_ID"
    DataSource    =  "Licensees"
    Height        =  285
    Left          =  8040
    TabIndex      =  8
    Top           =  1080
    Width         =  495
  End
  Begin TextBox LicenseeID
    Height        =  495
    Left          =  1440
    TabIndex      =  0
    Top           =  120
    Width         =  2775
  End
  Begin Data Licensees
    Caption       =  "Licensees"
    Connect       =  ""
    DatabaseName  =  "C:\RATFILES\THESIS\TEST_BED\VB\CSTS\CSTS.MDB"
    Exclusive     =  0  'False
    Height        =  495
    Left          =  5880
    Options       =  0
    ReadOnly      =  0  'False
    RecordSource  =  "select * from Licensees, Companies, Licensees INNER JOIN Companies ON
         Licensees.CMPY_LOC_ID = Companies.CMPY_LOC_ID order by LICENSEE_ID"
    Top           =  120
    Width         =  2175
  End
  Begin TextBox CompanyIDPK
    DataField     =  "Companies.CMPY_LOC_ID"
    DataSource    =  "Licensees"
    Height        =  285
    Left          =  8040
    TabIndex      =  11
    Top           =  5760
    Width         =  495
  End
  Begin Label lblLicID
    Caption       =  "Licensee ID"
    Height        =  255
    Left          =  120
    TabIndex      =  12
    Top           =  240
    Width         =  1455
  End
  Begin Menu MenuFile
    Caption       =  "&File"
    Begin Menu MenuFileExit
```

```
        Caption         =   "E&xit"
      End
    End
    Begin Menu MenuMaint
      Caption         =   "&Maintenance"
      Begin Menu MenuMaintDelCo
        Caption         =   "Delete &Company"
      End
      Begin Menu MenuMaintDelLic
        Caption         =   "Delete &Licensee"
      End
    End
  End
```

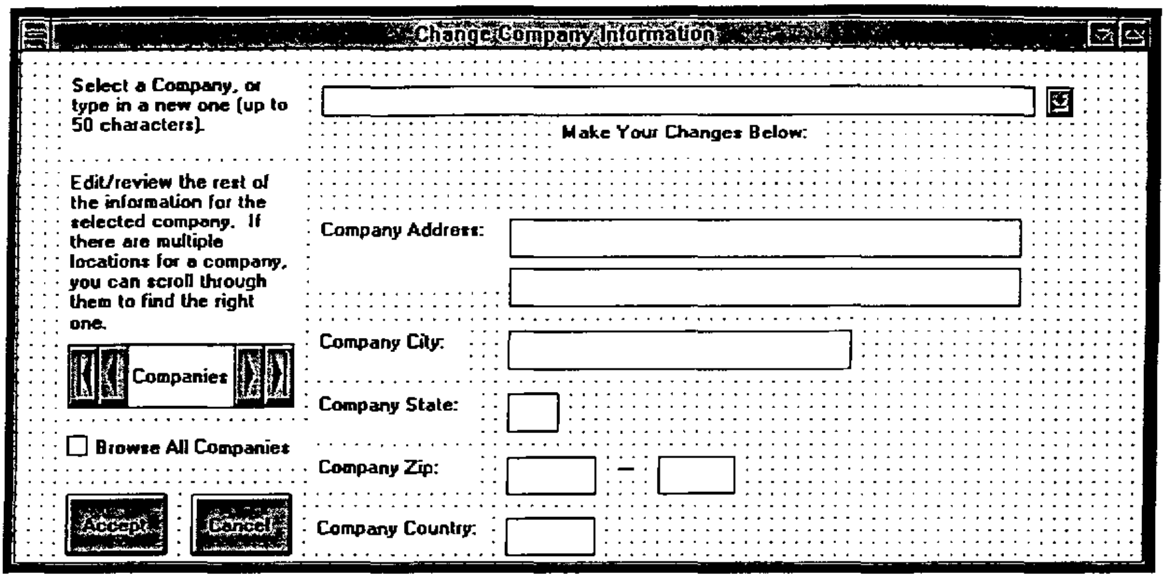
Main Form Object Behaviors (CSTS.FRM)

```
Option Explicit

Sub btnChgCoInfo_Click ()

  Dim SavePlace As Variant
  'SavePlace = Licensees.Recordset.Bookmark

  'Licensees.Recordset.AddNew

  ChangeCompany.Show 1

  'If LicenseeID.Text = "" Then
  '   Licensees.Recordset.Bookmark = SavePlace
  '   Exit Sub
  'End If

  SavePlace = Licensees.Recordset("LICENSEE_ID") 'LicenseeID.Text
  On Error GoTo CheckErr

  Licensees.Recordset.Update
  Licensees.Refresh
  Licensees.Recordset.FindFirst "LICENSEE_ID = '" & SavePlace & "'"

  Exit Sub 'No errors

CheckErr:
  Dim msg As String
  Dim Answer As Integer

  Select Case Err
    Case 3022
      msg = "That License ID already exists.  Click Yes if you want to go to that record, or Click No if
                you want to try again."
```

```
         Answer = MsgBox(msg, 4, "Duplicate ID Decision")
         If Answer = 6 Then 'Yes, go to existing record
            SavePlace = Licensees.Recordset("LICENSEE_ID") 'LicenseeID.Text
            Licensees.Recordset.FindFirst "LICENSEE_ID = '" & SavePlace & "'"
            Exit Sub
         Else
            'btnNewLic_Click  'No, try again
            Exit Sub
         End If

      Case 3058
         msg = "You must choose a Company affiliation for consistency's sake.  Please try again."
         MsgBox msg
         btnChgCoInfo_Click

      Case 3101
         msg = "You must choose a Company affiliation for consistency's sake.  Please try again."
         MsgBox msg
         btnChgCoInfo_Click

   End Select

   Resume

End Sub

Sub btnCoMaint_Click ()

   CSTSMnt.Show 1

End Sub

Sub btnSprtCall_Click ()

   SupportCall.Show 1

End Sub

Sub LicenseeID_LostFocus ()

   Dim SavePlace As Variant
   Dim SaveAffil As Variant
   Dim SQL As String
   Dim CompID As Integer

   SavePlace = LicenseeID.Text
   SaveAffil = Licensees.Recordset("Licensees.CMPY_LOC_ID")

   If SavePlace = "" Then
      Licensees.Recordset.MovePrevious
      Licensees.Recordset.MoveNext
      Exit Sub
   End If
```

```
    Licensees.Recordset.FindFirst "LICENSEE_ID = '" & SavePlace & "'"

    'Focus will now be on desired record IF it exists

    If Licensees.Recordset.NoMatch = True Then

      'Create a new LICENSEE record.
      Licensees.Recordset.AddNew
      Licensees.Recordset("LICENSEE_ID") = SavePlace
      'Each new Licensee must have a Company Affiliation or else the JOIN will be broken
      Licensees.Recordset("Licensees.CMPY_LOC_ID") = SaveAffil 'Set a Default
      Licensees.Recordset.Update
      'Set current record to this new one
      Licensees.Recordset.FindFirst "LICENSEE_ID = '" & SavePlace & "'"

      'Automatically invoke the Company info form
      btnChgCoInfo_Click

      'Refresh with all the lastest info
      Licensees.Recordset.Update
      Licensees.Refresh
      Licensees.Recordset.FindFirst "LICENSEE_ID = '" & SavePlace & "'"

   End If

 End Sub

 Sub Licensees_Error (DataErr As Integer, response As Integer)
   Dim msg As String
   Dim Answer As Integer
   Dim SavePlace As Variant
   Select Case DataErr
     Case 3022
       msg = "That License ID already exists.  Click Yes if you want to go to that record, or Click No if
                 you want to try again."
       Answer = MsgBox(msg, 4, "Duplicate ID Decision")
       If Answer = 6 Then 'Yes, go to existing record
         SavePlace = LicenseeID.Text
         Licensees.Recordset.FindFirst "LICENSEE_ID = '" & SavePlace & "'"
       Else
         'btnNewLic_Click  'No, try again
       End If
       response = 0

   End Select
End Sub

Sub Licensees_Reposition ()

   On Error GoTo CheckError

   LicenseeID.Text = Licensees.Recordset("LICENSEE_ID")
```

```
    Exit Sub

  CheckError:
    Dim msg As String
    Dim Answer As Integer
    Dim SavePlace As Variant

    Select Case Err
      Case 3022
        msg = "That License ID already exists.  Click Yes if you want to go to that record, or Click No if
                you want to try again."
        Answer = MsgBox(msg, 4, "Duplicate ID Decision")
        If Answer = 6 Then 'Yes, go to existing record
          SavePlace = LicenseeID.Text
          Licensees.Recordset.FindFirst "LICENSEE_ID = '" & SavePlace & "'"
          Exit Sub
        Else
          'btnNewLic_Click  'No, try again
          Exit Sub
        End If

      Case 3058
        msg = "You must choose a Company affiliation for consistency's sake.  Please try again."
        MsgBox msg
        'btnNewLic_Click

      Case 94
        'btnChgCoInfo_Click
        Exit Sub
    End Select

    Resume

End Sub

Sub MenuFileExit_Click ()

    End

End Sub

Sub MenuMaintDelCo_Click ()

    CSTSMnt.Show 1

End Sub

Sub MenuMaintDelLic_Click ()
Licensees.Recordset.Delete
Licensees.Recordset.MoveNext

End Sub
```



Select a Company, or type in a new one (up to 50 characters).

Make Your Changes Below:

Edit/review the rest of the information for the selected company. If there are multiple locations for a company, you can scroll through them to find the right one.

Customer Support Tracking System Company Maintenance Screen
(Visual Basic Version)

Company Maintenance Form Object Definitions (CHANGECO.FRM)

```
VERSION 2.00
Begin Form ChangeCompany
  Caption        =  "Change Company Information"
  ClientHeight   =  4995
  ClientLeft     =  75
  ClientTop      =  2100
  ClientWidth    =  10665
  Height         =  5400
  Left           =  15
  LinkTopic      =  "Form1"
  ScaleHeight    =  4995
  ScaleWidth     =  10665
  Top            =  1755
  Width          =  10785
  Begin CheckBox chkBrowser
    Caption      =  "Browse All Companies"
    Height       =  255
    Left         =  480
    TabIndex     =  18
    Top          =  3720
    Width        =  2295
```

```
End
Begin CommandButton btnCancel
  Caption       =  "Cancel"
  Height        =  615
  Left          =  1680
  TabIndex      =  9
  Top           =  4320
  Width         =  975
End
Begin TextBox tedCompanyCntry
  DataField     =  "CMPY_CNTRY"
  DataSource    =  "Companies"
  Height        =  375
  Left          =  4680
  TabIndex      =  7
  Top           =  4560
  Width         =  855
End
Begin TextBox CompanyZPExt
  DataField     =  "CMPY_EXT_ZIP"
  DataSource    =  "Companies"
  Height        =  375
  Left          =  6120
  TabIndex      =  6
  Top           =  3960
  Width         =  735
End
Begin TextBox tedCompanySt
  DataField     =  "CMPY_ST_CD"
  DataSource    =  "Companies"
  Height        =  375
  Left          =  4680
  TabIndex      =  4
  Top           =  3360
  Width         =  495
End
Begin TextBox tedCompanyCity
  DataField     =  "CMPY_CITY_NM"
  DataSource    =  "Companies"
  Height        =  375
  Left          =  4680
  TabIndex      =  3
  Top           =  2760
  Width         =  3255
End
Begin TextBox tedCompanyAddr2
  DataField     =  "CMPY_AD2"
  DataSource    =  "Companies"
  Height        =  375
  Left          =  4680
  TabIndex      =  2
  Top           =  2160
  Width         =  4815
```

```
End
Begin TextBox tedCompanyAddr1
   DataField       =   "CMPY_AD1"
   DataSource      =   "Companies"
   Height          =   375
   Left            =   4680
   TabIndex        =   1
   Top             =   1680
   Width           =   4815
End
Begin TextBox CompanyZip
   DataField       =   "CMPY_ZP"
   DataSource      =   "Companies"
   Height          =   375
   Left            =   4680
   TabIndex        =   5
   Top             =   3960
   Width           =   855
End
Begin ComboBox TempCoName
   Height          =   300
   Left            =   2880
   Sorted          =   -1  'True
   TabIndex        =   0
   Top             =   360
   Width           =   7095
End
Begin CommandButton btnAccept
   Caption         =   "Accept"
   Height          =   615
   Left            =   480
   TabIndex        =   8
   Top             =   4320
   Width           =   975
End
Begin Data Companies
   Caption         =   "Companies"
   Connect         =   ""
   DatabaseName    =   "C:\RATFILES\THESIS\TEST_BED\VB\CSTS\CSTS.MDB"
   Exclusive       =   0   'False
   Height          =   615
   Left            =   480
   Options         =   0
   ReadOnly        =   0   'False
   RecordSource    =   "Companies"
   Top             =   2880
   Width           =   2175
End
Begin Label lblCmpyCntry
   Caption         =   "Company Country:"
   Height          =   255
   Left            =   2880
   TabIndex        =   17
```

```
      Top          =  4560
      Width        =  1575
   End
   Begin Line Line5
      BorderWidth  =  2
      X1           =  5760
      X2           =  5880
      Y1           =  4080
      Y2           =  4080
   End
   Begin Label lblCmpyCity
      Caption      =  "Company City:"
      Height       =  375
      Left         =  2880
      TabIndex     =  14
      Top          =  2760
      Width        =  1335
   End
   Begin Label lblCmpyAd1
      Caption      =  "Company Address:"
      Height       =  615
      Left         =  2880
      TabIndex     =  16
      Top          =  1680
      Width        =  1695
   End
   Begin Label lblCmpySt
      Caption      =  "Company State:"
      Height       =  375
      Left         =  2880
      TabIndex     =  15
      Top          =  3360
      Width        =  1455
   End
   Begin Label lblNameChg
      Caption      =  "Make Your Changes Below:"
      Height       =  255
      Left         =  5160
      TabIndex     =  13
      Top          =  720
      Visible      =  0   'False
      Width        =  2415
   End
   Begin Label lblCmpyZP
      Caption      =  "Company Zip:"
      Height       =  255
      Left         =  2880
      TabIndex     =  12
      Top          =  3960
      Width        =  1455
   End
   Begin Label Label4
      Caption  =  "Edit/review the rest of the information for the selected company.  If there are
```

```
                  multiple locations for a company, you can scroll through them to find the right one."
      Height         =  1575
      Left           =  480
      TabIndex       =  11
      Top            =  1200
      Width          =  2175
   End
   Begin Label Label2
      Caption        =  "Select a Company, or type in a new one (up to 50 characters)."
      Height         =  735
      Left           =  480
      TabIndex       =  10
      Top            =  240
      Width          =  2175
   End
End
```

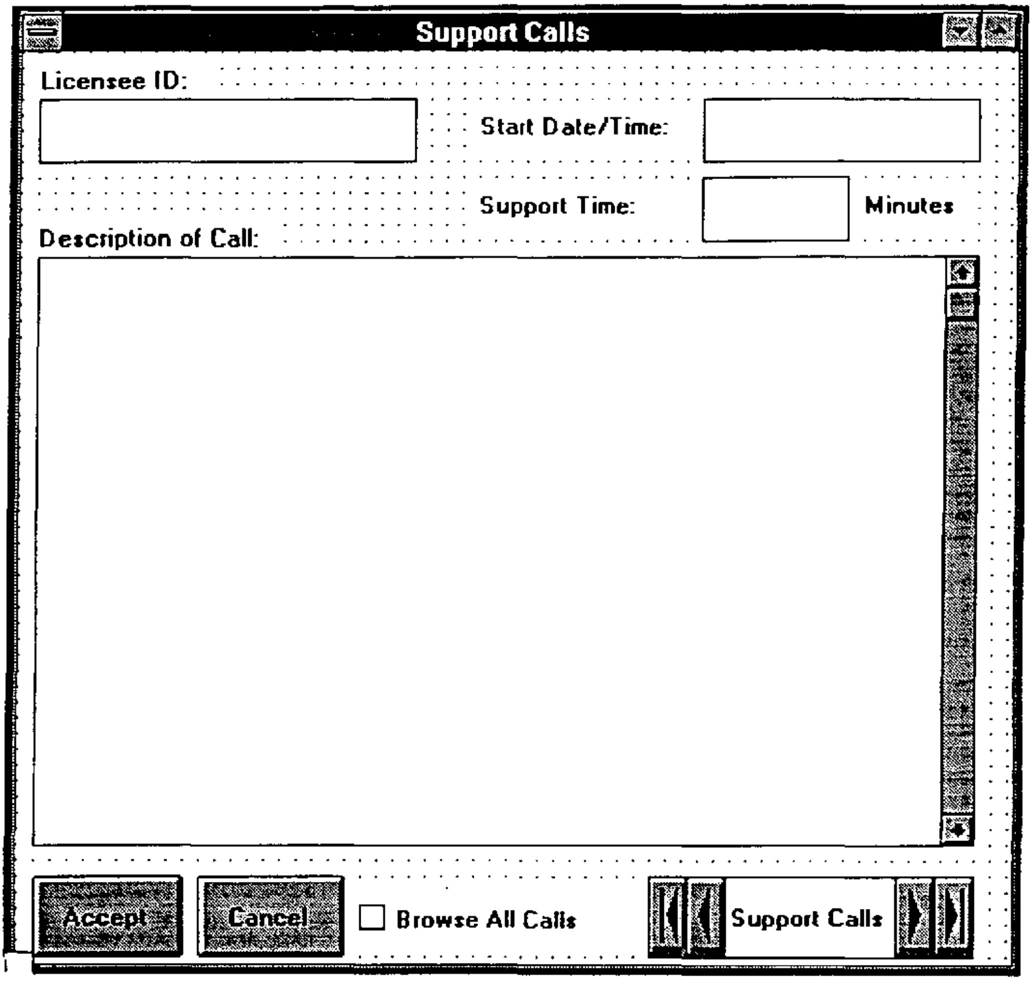
Company Maintenance Form Object Behaviors (CHANGECO.FRM)

```
Option Explicit

  Dim Loading As Integer

  Dim browsing As Integer

Sub btnAccept_Click ()

   Dim SQL As String
   Dim CompID As Integer
   Dim SavePlace As String

   'Mustn't have a blank company name, so let the default ride
   If TempCoName.Text = "" Then
      Unload ChangeCompany
      Exit Sub
   End If

   'Update the record with the current info
   Companies.Recordset.Edit
   Companies.Recordset("CMPY_AD1") = (tedCompanyAddr1.Text)
   Companies.Recordset("CMPY_AD2") = (tedCompanyAddr2.Text)
   Companies.Recordset("CMPY_CITY_NM") = (tedCompanyCity.Text)
   Companies.Recordset("CMPY_ST_CD") = (tedCompanySt.Text)
   Companies.Recordset("CMPY_EXT_ZIP") = Val(CompanyZPExt.Text)
   Companies.Recordset("CMPY_ZP") = Val(CompanyZip.Text)
   Companies.Recordset("CMPY_CNTRY") = (tedCompanyCntry.Text)
   CSTSMain.CompanyIDFK = Companies.Recordset("CMPY_LOC_ID")
   Companies.Recordset.Update

   Unload ChangeCompany

End Sub
```

```
 Sub btnCancel_Click ()

    Unload ChangeCompany
    'NOTE: if you move the record with a button, any edits will be committed!

 End Sub

 Sub chkBrowser_Click ()

    If chkBrowser.Value = 1 Then

       browsing = True
       TempCoName_Click
    Else

       browsing = False
       TempCoName_Click

    End If

 End Sub

 Sub Companies_Reposition ()

'The purpose of this procedure is to keep the company name synchronized with
'the rest of the record when browsing, since it is not directly linked to the
'table.

    'If we are populating the TempCoName menu (or we know that the current
    'record will be NULL), then we want to exit this procedure
    If Loading Then
       Exit Sub
    End If

    'Otherwise, set the box to the value of the current record
    TempCoName.Text = Companies.Recordset("CMPY_NM")

End Sub

Sub Form_Load ()

    Dim SQL As String
    Dim PrevLoc As Integer
    SQL = "select * from Companies order by CMPY_NM"
    Companies.RecordSource = SQL
    Companies.Refresh

    Loading = True  'Flag for Reposition Event

    'Populate the pull-down menu
    TempCoName.AddItem "<Browse All Companies>"
    Do While Not Companies.Recordset.EOF
       'Skip duplicate names
```

```
      If TempCoName.List(TempCoName.NewIndex) <> Companies.Recordset("CMPY_NM") Then
         TempCoName.AddItem Companies.Recordset("CMPY_NM")
      End If

      Companies.Recordset.MoveNext

   Loop

   'Synchronize the Companies record with the Licensees record
   PrevLoc = CSTSMain.Licensees.Recordset("Licensees.CMPY_LOC_ID")
   Companies.Recordset.FindFirst "CMPY_LOC_ID = " & PrevLoc & ""
   TempCoName.Text = Companies.Recordset("CMPY_NM")

   'The work for the rest of the Company data is the same as for a click
   TempCoName_Click

   Loading = False  'Flag for Reposition Event

End Sub

Sub TempCoName_Click ()

   Dim SavePlace As Variant
   Dim SaveIndex As Variant
   Dim SQL As String
   Dim CompID As Integer

   'Hang on to the desired company name
   SavePlace = TempCoName.Text

   'If it's the same as the current record, just hang on to the ID
   If SavePlace = Companies.Recordset("CMPY_NM") Then
      CompID = Companies.Recordset("CMPY_LOC_ID")
   Else
      'Otherwise, move to the beginning of the new name, and grab the ID
      Companies.Recordset.FindFirst "CMPY_NM = '" & SavePlace & "'"
      CompID = Companies.Recordset("CMPY_LOC_ID")
   End If

   'Make sure that the entire table is available for the upcoming FindFirst
   'and sort it by company name, which is more meaningful to users than ID
   SQL = "select * from Companies order by CMPY_NM"
   Companies.RecordSource = SQL
   Companies.Refresh

   'Our job is done if the user is wishing to browse all companies
   If SavePlace = "<Browse All Companies>" Then
      chkBrowser.Value = 1
      browsing = True
      Exit Sub
   End If

   'Now match the User-Supplied Company Name
```

```
    Companies.Recordset.FindFirst "CMPY_NM = '" & SavePlace & "'"

    If Companies.Recordset.NoMatch = True Then
      'Since there is no match, create a new COM_LOC_ID and add a new record with new company
          name.
      Loading = True 'to avoid illegal null in Companies_Reposition
      'move to the highest numbered company ID
      SQL = "select * from Companies order by CMPY_LOC_ID"
      Companies.RecordSource = SQL
      Companies.Refresh
      Companies.Recordset.MoveLast
      'and increment it to set the ID for the new company
      CompID = Companies.Recordset("CMPY_LOC_ID")
      CompID = CompID + 1
      'Then create a new record with the new name and ID
      Companies.Recordset.AddNew
      Companies.Recordset("CMPY_LOC_ID") = CompID
      Companies.Recordset("CMPY_NM") = SavePlace
      Companies.Recordset.Update
      Companies.Recordset.MoveLast
      'Add the new company to the pull-down menu
      TempCoName.AddItem SavePlace'or...Companies.Recordset("CMPY_NM")
      'Finally, resort the table and put the new record in front of the user
      TempCoName.Text = Companies.Recordset("CMPY_NM")
      SQL = "select * from Companies order by CMPY_NM"
      Companies.RecordSource = SQL
      Companies.Refresh
      Companies.Recordset.FindFirst "CMPY_LOC_ID = " & CompID
      Loading = False

   Else
      'Since there is a match, filter the records and go to the first one
      'Unless the user is in Browsing Mode
      If browsing Then
        Companies.Recordset.FindFirst "CMPY_LOC_ID = " & Str(CompID)
        Exit Sub
      End If

      SQL = "select * from Companies where CMPY_NM = '" & TempCoName.Text & "'"
      Companies.RecordSource = SQL
      Companies.Refresh
      Companies.Recordset.MoveLast
      'Move to the most recent ID
      Companies.Recordset.FindFirst "CMPY_LOC_ID = " & Str(CompID)

  End If
End Sub

Sub TempCoName_LostFocus ()

  TempCoName_Click

End Sub
```



Customer Support Tracking System Support Call Screen (Visual Basic Version)

Support Call Form Object Definitions (SUPPORTC.FRM)

```
VERSION 2.00
Begin Form SupportCall
   Caption         =   "Support Calls"
   ClientHeight    =   6900
   ClientLeft      =   1320
   ClientTop       =   1815
   ClientWidth     =   7365
   Height          =   7305
   Left            =   1260
   LinkTopic       =   "Form1"
   ScaleHeight     =   6900
   ScaleWidth      =   7365
```

```
Top          =  1470
Width        =  7485
Begin CheckBox chkBrowser
  Caption        =  "Browse All Calls"
  Enabled        =  0   'False
  Height         =  375
  Left           =  2520
  TabIndex       =  6
  Top            =  6360
  Width          =  2055
End
Begin TextBox tedLicID
  DataField      =  "LICENSEE_ID"
  DataSource     =  "SupportCalls"
  Height         =  495
  Left           =  120
  TabIndex       =  3
  Top            =  360
  Width          =  2775
End
Begin CommandButton btnCancel
  Caption        =  "Cancel"
  Height         =  615
  Left           =  1320
  TabIndex       =  2
  Top            =  6240
  Width          =  1095
End
Begin CommandButton btnAccept
  Caption        =  "Accept"
  Height         =  615
  Left           =  120
  TabIndex       =  1
  Top            =  6240
  Width          =  1095
End
Begin Data SupportCalls
  Caption        =  "Support Calls"
  Connect        =  ""
  DatabaseName   =  "C:\RATFILES\THESIS\TEST_BED\VB\CSTS\CSTS.MDB"
  Exclusive      =  0   'False
  Height         =  615
  Left           =  4680
  Options        =  0
  ReadOnly       =  0   'False
  RecordSource   =  "Support"
  Top            =  6240
  Width          =  2415
End
Begin TextBox tedDescription
  DataField      =  "SPRT_CMT"
  DataSource     =  "SupportCalls"
  Height         =  4455
```

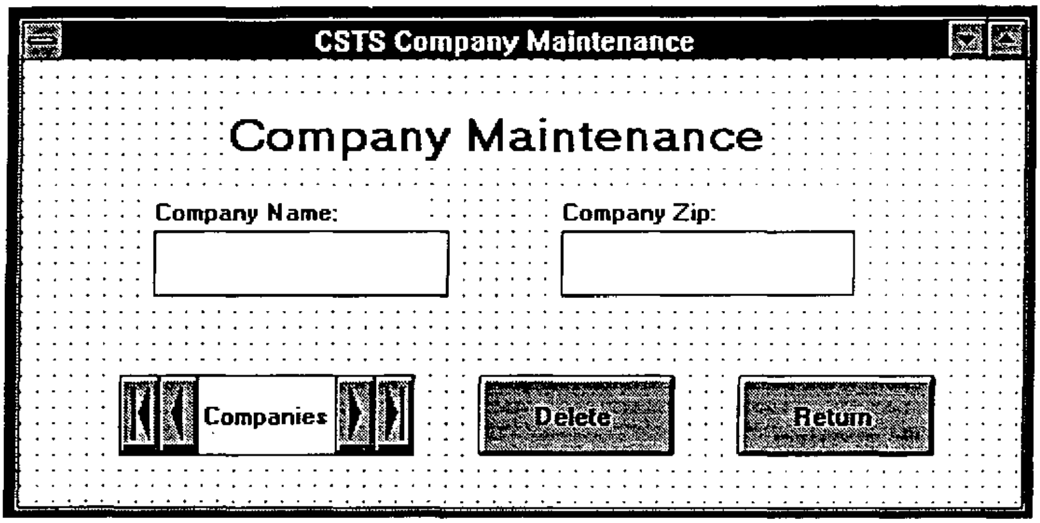
```
   Left          =  120
   MultiLine      =  -1  'True
   ScrollBars     =  2  'Vertical
   TabIndex       =  0
   Top           =  1560
   Width          =  6975
End
Begin Label lblSprtTm
   Caption        =  "Support Time:"
   Height         =  255
   Left          =  3360
   TabIndex       =  11
   Top           =  1080
   Width          =  1575
End
Begin Label lblStrtDt
   Caption        =  "Start Date/Time:"
   Height         =  255
   Left          =  3360
   TabIndex       =  10
   Top           =  480
   Width          =  1575
End
Begin Label lblMin
   Caption        =  "Minutes"
   Height         =  255
   Left          =  6240
   TabIndex       =  9
   Top           =  1080
   Width          =  735
End
Begin Label lblCallDescr
   Caption        =  "Description of Call:"
   Height         =  255
   Left          =  120
   TabIndex       =  8
   Top           =  1320
   Width          =  1695
End
Begin Label lblLicID
   Caption        =  "Licensee ID:"
   Height         =  255
   Left          =  120
   TabIndex       =  7
   Top           =  120
   Width          =  1215
End
Begin Label lblStartTime
   BorderStyle    =  1  'Fixed Single
   DataField      =  "SPRT_DATE"
   DataSource     =  "SupportCalls"
   Height         =  495
   Left          =  5040
```

```
      TabIndex      = 5
      Top           = 360
      Width         = 2055
   End
   Begin Label lblSprtTimeUsed
      BorderStyle   = 1 'Fixed Single
      DataField     = "SPRT_TM"
      DataSource    = "SupportCalls"
      Height        = 495
      Left          = 5040
      TabIndex      = 4
      Top           = 960
      Width         = 1095
   End
End
```

Support Call Form Object Behaviors (SUPPORTC.FRM)

```
Option Explicit

Dim Prevloc As String

Sub btnAccept_Click ()

    Dim Duration As Integer

    'Calculate the time used for the call
    Duration = DateDiff("n", lblStartTime.Caption, Now)
    Duration = Duration / 15
    Duration = Duration * 15 + 15
    lblSprtTimeUsed.Caption = Duration

    'Put the record pointer at the new record
    SupportCalls.Recordset.MoveLast

    'Update the record with the current info
    SupportCalls.Recordset.Edit
    SupportCalls.Recordset("SPRT_CMT") = (tedDescription.Text)
    SupportCalls.Recordset("SPRT_TM") = Val(lblSprtTimeUsed.Caption)
    SupportCalls.Recordset.Update

    Unload SupportCall

End Sub

Sub btnCancel_Click ()

    'Put the record pointer at the new record
    SupportCalls.Recordset.MoveLast

    'Delete the newly created record
    SupportCalls.Recordset.Delete
    SupportCalls.Recordset.MoveNext

    Unload SupportCall
    'NOTE: if you move the record with a button, any edits will be committed!

 End Sub

 Sub Form_Load ()

    Dim SQL As String
    Static Prevloc As String

    Prevloc = CSTSMain.Licensees.Recordset("Licensees.LICENSEE_ID")

    'Filter the records to the selected Licensee
    SQL = "select * from Support where Support.LICENSEE_ID = """
    SQL = SQL & Prevloc
    SQL = SQL & """ order by SPRT_DATE"
    SupportCalls.RecordSource = SQL
    SupportCalls.Refresh

    'Create a new record
    SupportCalls.Recordset.AddNew

    'Set the start date/time (key) and ID for the new support call
    SupportCalls.Recordset("SPRT_DATE") = Now
    SupportCalls.Recordset("LICENSEE_ID") = Prevloc
    SupportCalls.Recordset("SPRT_TM") = 0 'For Null protection, (just in case)
    SupportCalls.Recordset.Update
    SupportCalls.Recordset.MoveLast

End Sub

Sub Form_Unload (Cancel As Integer)

   Dim Total As Integer
   Dim SQL As String
   'Static PrevLoc As String

   're-filter, in case the user has selected "Browse All Calls"
   SQL = "select * from Support where Support.LICENSEE_ID = """ & Prevloc & """"
   SupportCalls.RecordSource = SQL
   SupportCalls.Refresh

   'Calculate the total support time used to date for that licensee
   Total = 0
   Do While Not SupportCalls.Recordset.EOF
      Total = Total + SupportCalls.Recordset("SPRT_TM")
      SupportCalls.Recordset.MoveNext

   Loop
```

```
    CSTSMain.Licensees.Recordset.Edit
    CSTSMain.Licensees.Recordset("LIC_TOT_SPRT_TM") = Total
    CSTSMain.Licensees.Recordset.Update

End Sub

Sub tedLicID_GotFocus ()

    tedDescription.SetFocus

End Sub
```



Customer Support Tracking System Company Deletion Screen (Visual Basic Version)

Company Deletion Form Object Definitions (CSTSMNT.FRM)

```
VERSION 2.00
Begin Form CSTSMnt
  Caption       =  "CSTS Company Maintenance"
  ClientHeight  =  3405
  ClientLeft    =  1095
  ClientTop     =  1485
  ClientWidth   =  7365
  Height        =  3810
  Left          =  1035
  LinkTopic     =  "Form1"
  ScaleHeight   =  3405
  ScaleWidth    =  7365
  Top           =  1140
  Width         =  7485
  Begin CommandButton Command2
    Caption       =  "Return"
    Height        =  615
    Left          =  5280
    TabIndex      =  1
    Top           =  2400
    Width         =  1455
  End
  Begin TextBox tedCompanyName
    DataField     =  "CMPY_NM"
    DataSource    =  "Companies"
    Height        =  495
    Left          =  960
```

```
      TabIndex      =  2
      TabStop       =  0  'False
      Top           =  1320
      Width         =  2175
   End
   Begin TextBox tedCompanyZip
      DataField     =  "CMPY_ZIP"
      DataSource    =  "Companies"
      Height        =  495
      Left          =  3960
      TabIndex      =  3
      TabStop       =  0  'False
      Top           =  1320
      Width         =  2175
   End
   Begin CommandButton Command1
      Caption       =  "Delete"
      Height        =  615
      Left          =  3360
      TabIndex      =  0
      Top           =  2400
      Width         =  1455
   End
   Begin Data Companies
      Caption       =  "Companies"
      Connect       =  ""
      DatabaseName  =  "C:\RATFILES\THESIS\TEST_BED\VB\CSTS\CSTS.MDB"
      Exclusive     =  0  'False
      Height        =  615
      Left          =  720
      Options       =  0
      ReadOnly      =  0  'False
      RecordSource  =  "Companies"
      Top           =  2400
      Width         =  2175
   End
   Begin Label Label3
      Alignment     =  2  'Center
      AutoSize      =  -1  'True
      Caption       =  "Company Maintenance"
      FontBold      =  -1  'True
      FontItalic    =  0  'False
      FontName      =  "MS Sans Serif"
      FontSize      =  18
      FontStrikethru =  0  'False
      FontUnderline  =  0  'False
      Height        =  435
      Left          =  1485
      TabIndex      =  6
      Top           =  360
      Width         =  4005
   End
   Begin Label Label2
```

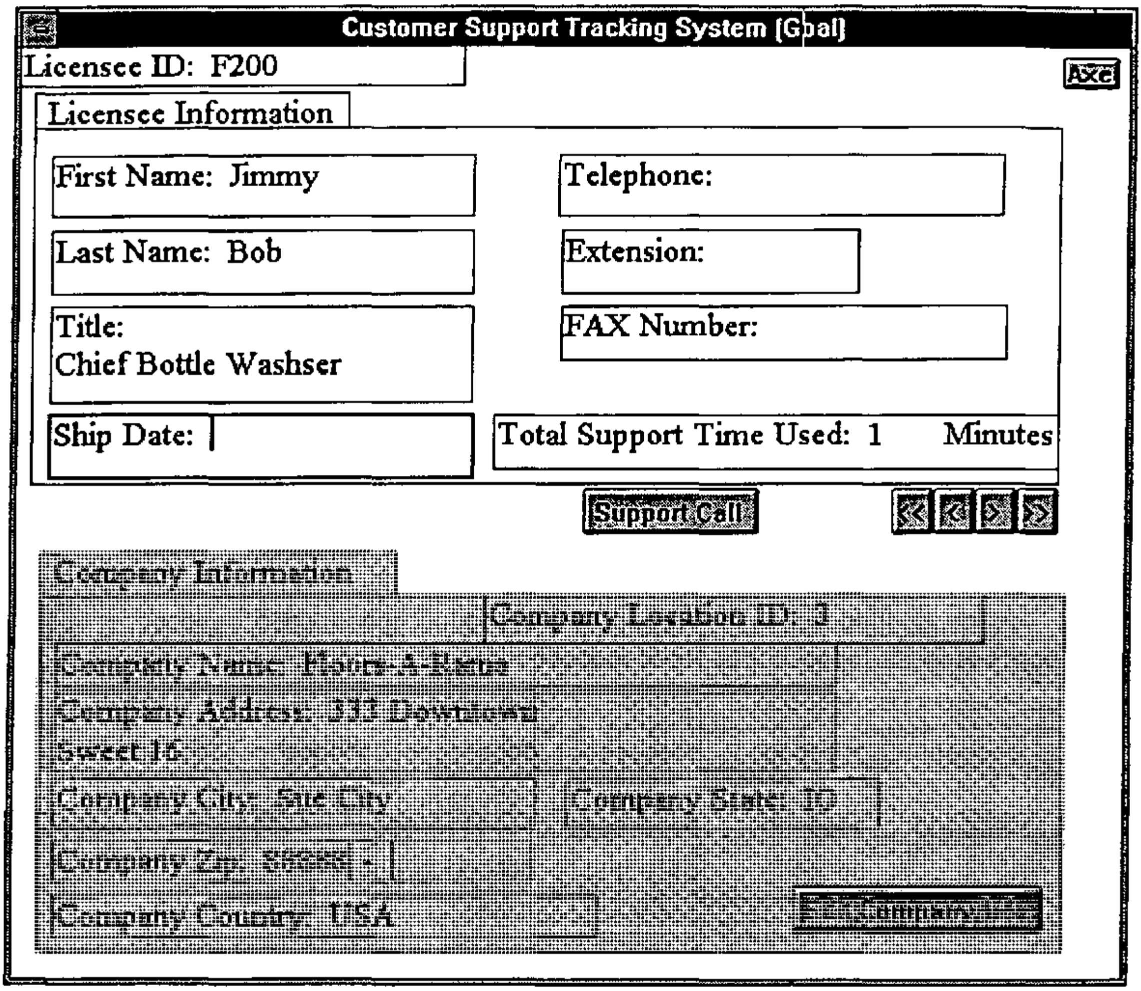
```
    Caption      =  "Company Zip:"
    Height       =  255
    Left       =  3960
    TabIndex      =  5
    Top        =  1080
    Width        =  2055
  End
  Begin Label Label1
    Caption      =  "Company Name:"
    Height       =  255
    Left       =  960
    TabIndex      =  4
    Top        =  1080
    Width        =  1935
  End
End
```

Company Deletion Form Object Behaviors (CSTSMNT.FRM)

```
Option Explicit

Sub Command1_Click ()
  Companies.Recordset.Delete
  Companies.Recordset.MoveNext
End Sub

Sub Command2_Click ()

   Unload CSTSMnt

End Sub
```

APPENDIX C

Customer Support Tracking System ObjectVision Listings/Screens



Licensee ID: F200

Licensee Information

First Name: Jimmy

Telephone:

Last Name: Bob

Extension:

Title:
Chief Bottle Washser

FAX Number:

Ship Date:

Total Support Time Used: 1 Minutes

Company Information

Company Location ID: 3

Company Name: Floors-A-Rama

Company Address: 333 Downtown
Sweet 16

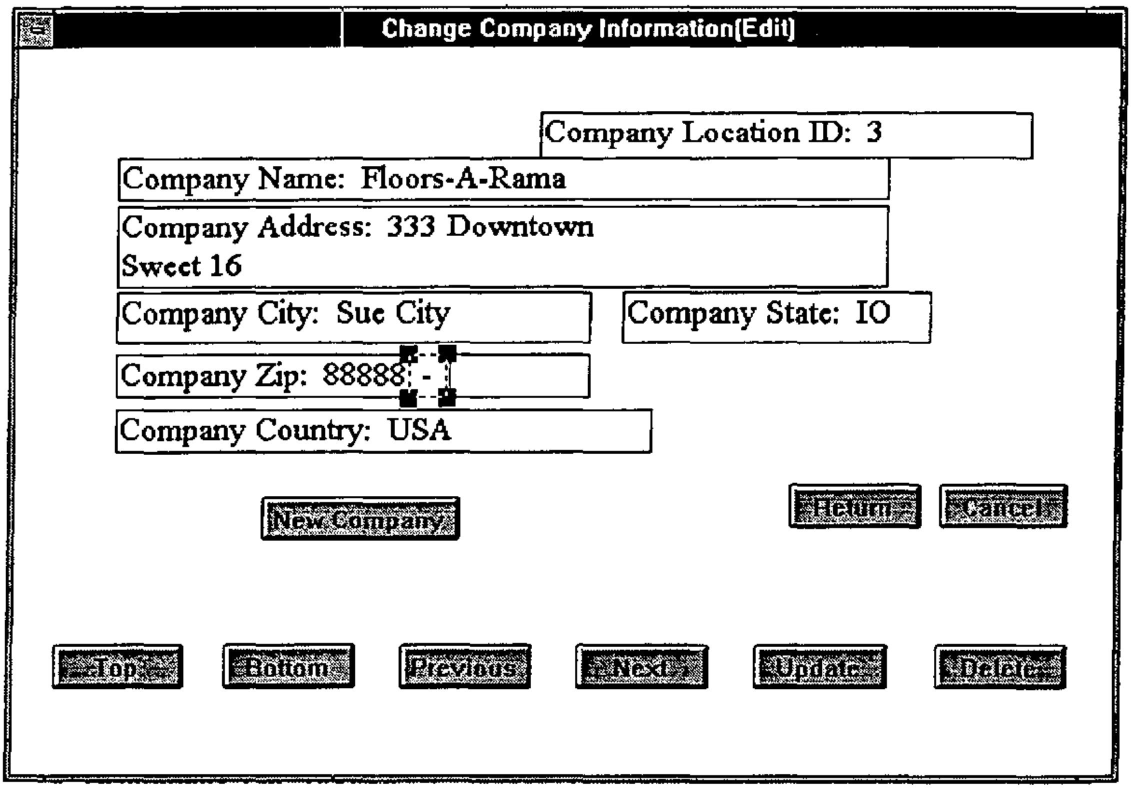
Company City: Sue City

Company State: ID

Company Zip: 55555 -

Company Country: USA

Customer Support Tracking System Main Screen (ObjectVision Version)



Change Company Information(Edit)

Company Location ID: 3

Company Name: Floors-A-Rama

Company Address: 333 Downtown
Sweet 16

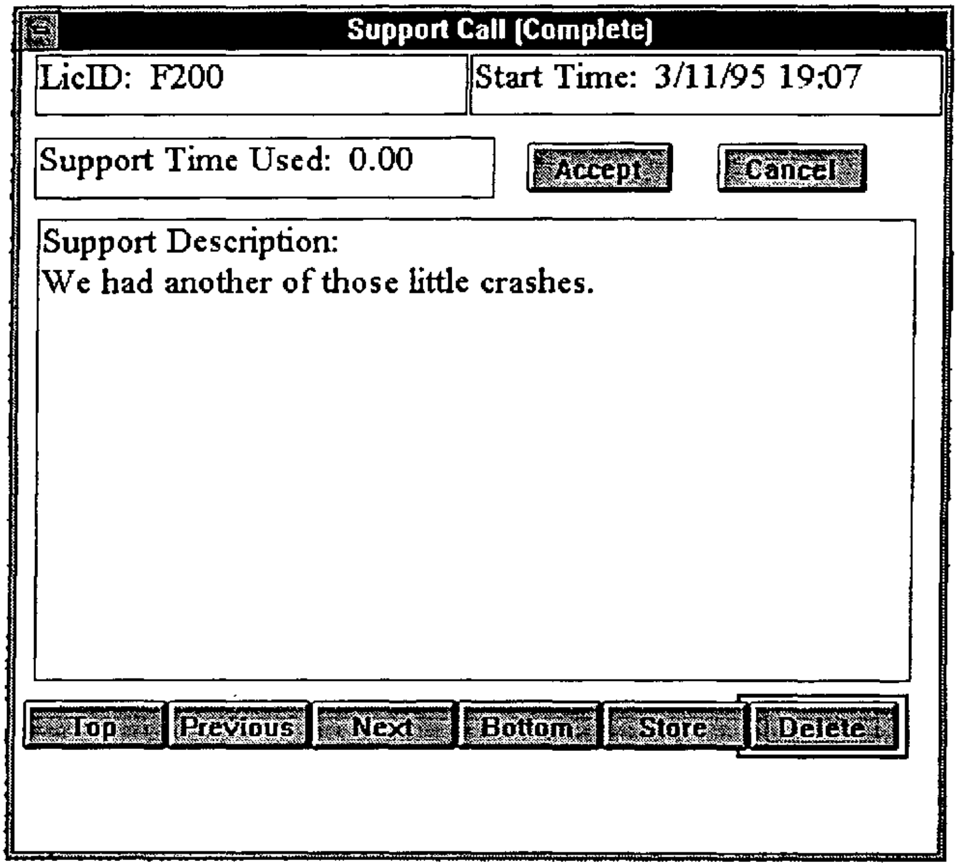
Company City: Sue City

Company State: IO

Company Zip: 88888 -

Company Country: USA

Customer Support Tracking System Company Maintenance Screen
(ObjectVision Version)



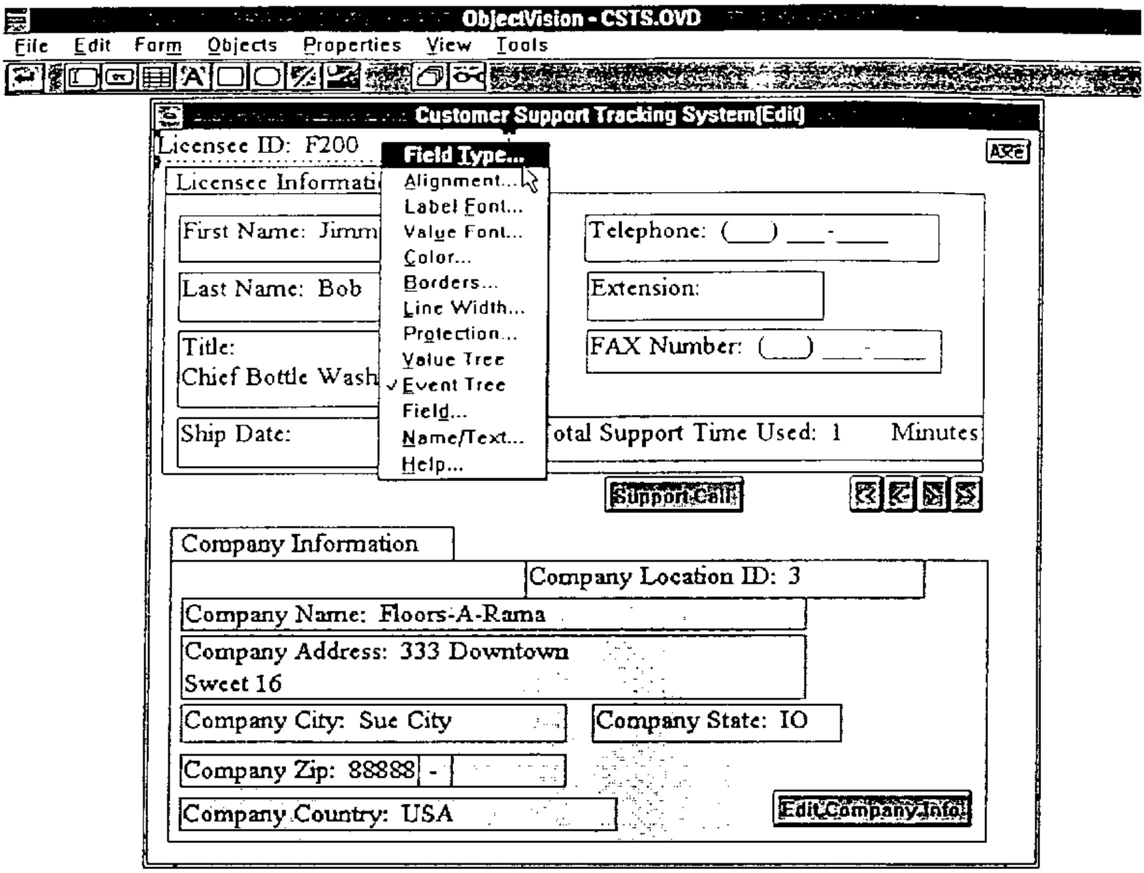
LicID: F200

Start Time: 3/11/95 19:07

Support Time Used: 0.00

Support Description:
We had another of those little crashes.

Customer Support Tracking System Support Call Screen (ObjectVision Version)



Licensee ID: F200

Licensee Informati

First Name: Jimm

Last Name: Bob

Title:
Chief Bottle Wash

Ship Date:

Telephone: (__) ___-____

Extension:

FAX Number: (__) ___-____

otal Support Time Used: 1 Minutes

Company Information

Company Location ID: 3

Company Name: Floors-A-Rama

Company Address: 333 Downtown
Sweet 16

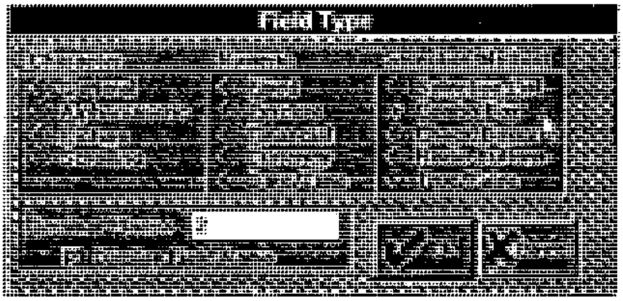
Company City: Sue City

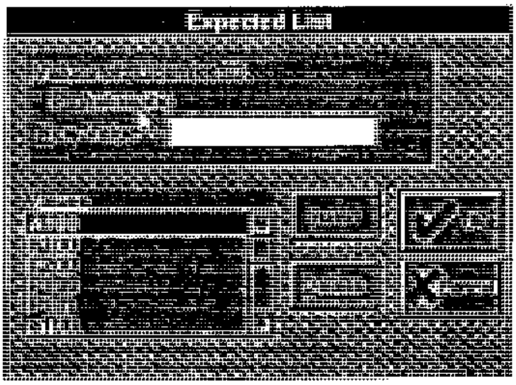
Company State: IO

Company Zip: 88888 -

Company Country: USA

1. Right-click on the field to bring up Attributes menu; Select "Field Type"

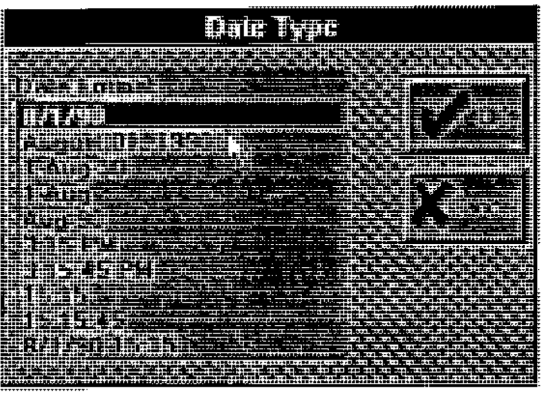




2. Select & OK "Combo Box"

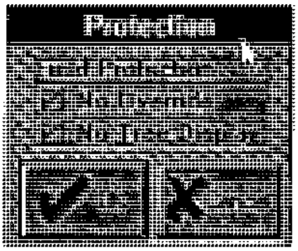
3. Select & OK "Automatic"; values will be populated from data base

Visual "Source Code" for a Typical Automatic Combo Box



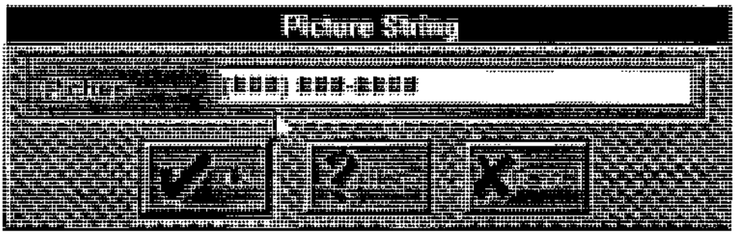
After Selecting "Field Type" from Attribute Menu, and Selecting "Date/Time" from the "Field Type" Dialog, Select & OK desired "Date Format"

Visual "Source Code" for a Typical Date Field



After Selecting "Protection" from Attribute Menu, Select & OK "No Override" and "No Tree Display"

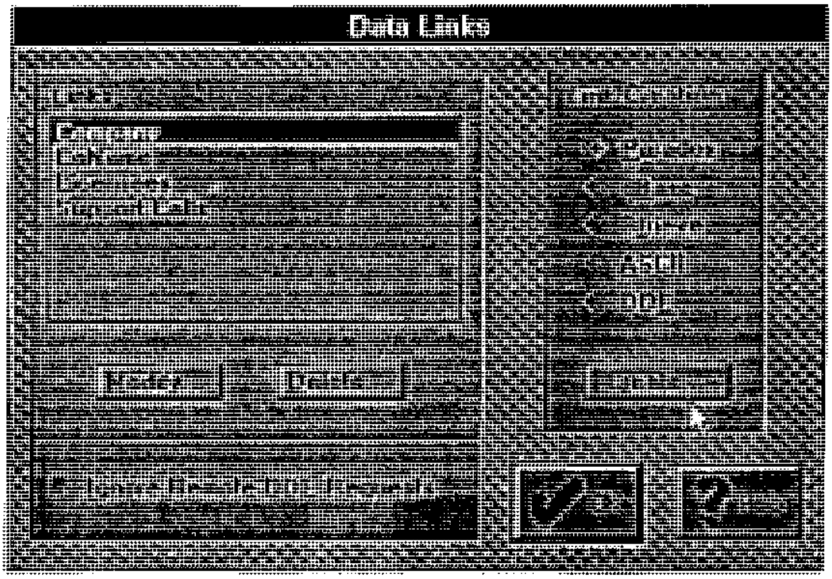
Visual "Source Code" for a Typical Protected (non-editable) Field



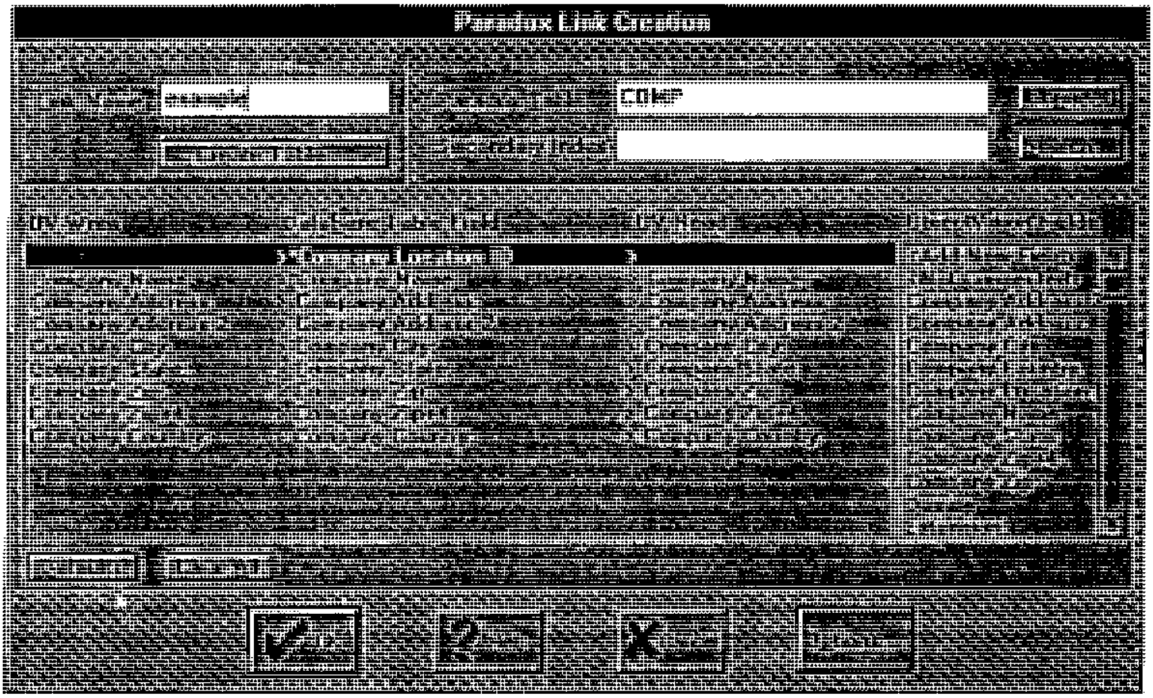
(###) ###-####

After Selecting "Field Type" from Attribute Menu, and Selecting "Picture" from the "Field Type" Dialog, Type in & OK desired "Picture String"

Visual "Source Code" for a Typical Picture (constrained) Field

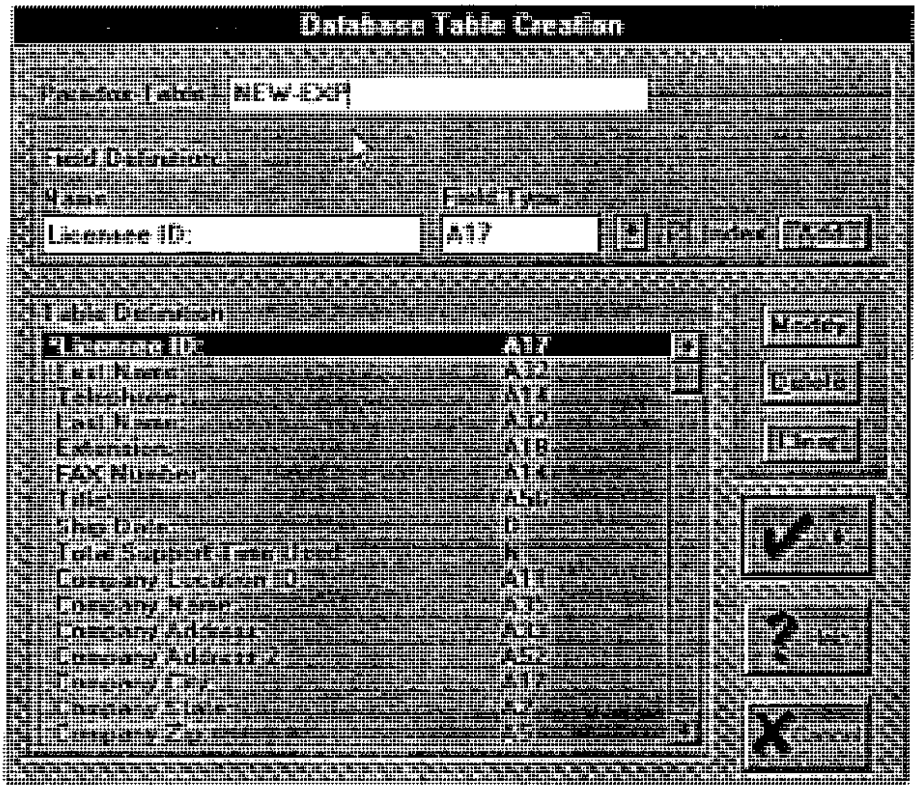


1. Open "Data Links" Tool, Select the desired Data Base Type, and Click on "Create..."



2. IF a Table already exists, Type in a new "Link Name" on the "Link Creation" Dialog, Type in (or "Search..." for) a Table, Click on "Defaults"; ObjectVision matches and links "Data Base Table Fields" with "ObjectVision Fields"; IF the Table must be created, Click on "Create Table..." and go on to step 3

Visual "Source Code" for Data Link Creation



Paradox Table: NEW-EXP

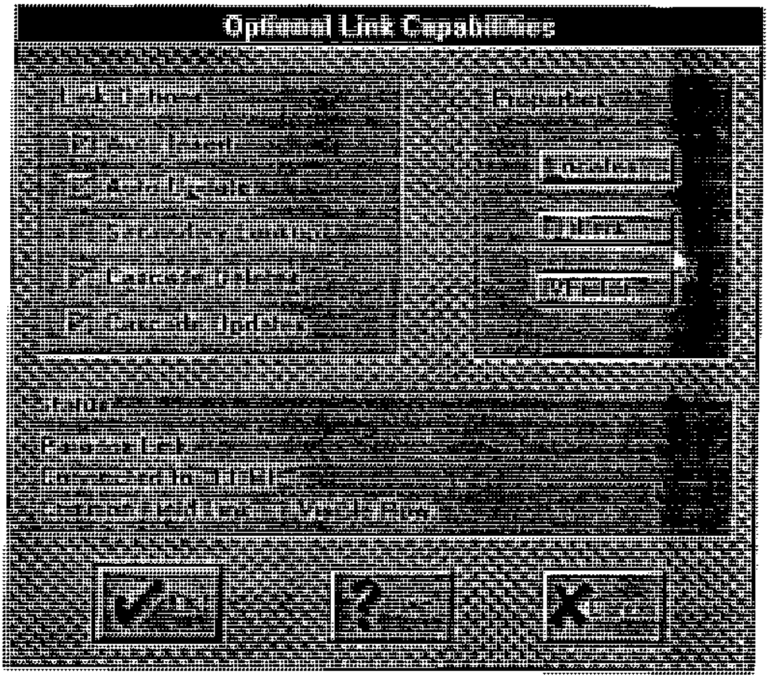
Name: Licensee ID:

Field Type: A17

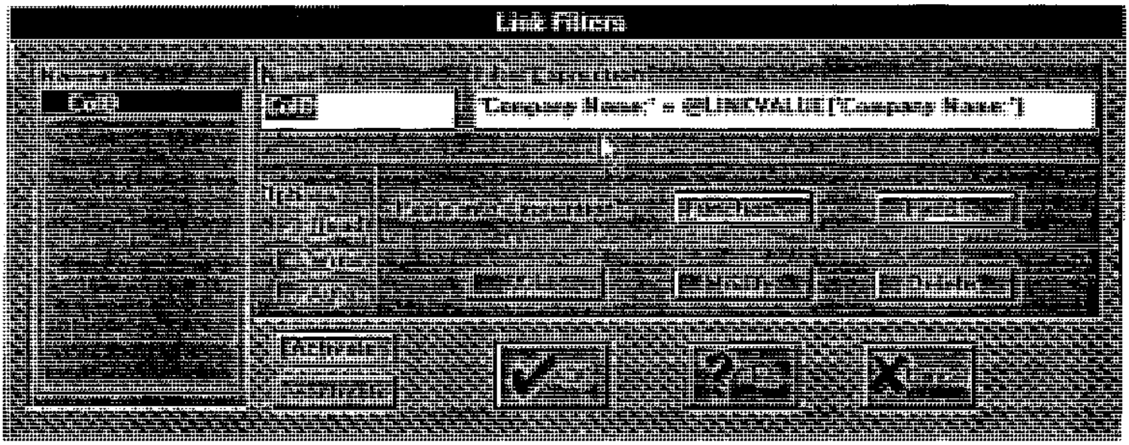
*Licensee ID:	A17
First Name	A32
Telephone	A14
Last Name	A32
Extension	A18
FAX Number	A14
Title	A50
Ship Date	D
Total Support Time Used	N
Company Location ID	A11
Company Name	A[illegible]
Company Address	A[illegible]
Company Address 2	A52
Company City	A17
Company State	A[illegible]
Company Zip	A[illegible]

3. On the "Data Base Table Creation" Dialog, Type in a new Table Name and then edit (or accept as-is) the "Table Definitions" automatically drafted by ObjectVision based on the user interface fields created to that point

Visual "Source Code" for Data Link Creation (continued)



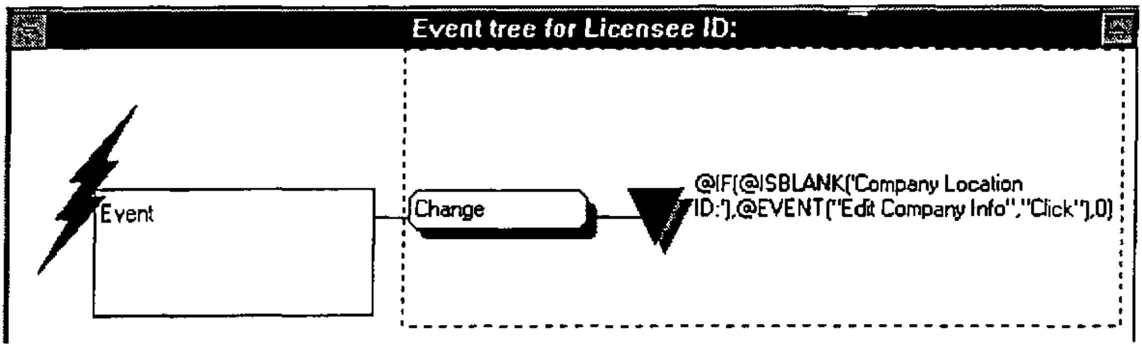
4. Once the link is OK'd, the "Optional Link Capabilities" Dialog appears for selecting (for example) Referential Integrity Rules and Filters



"Company Name" = @LINKVALUE("Company Name")

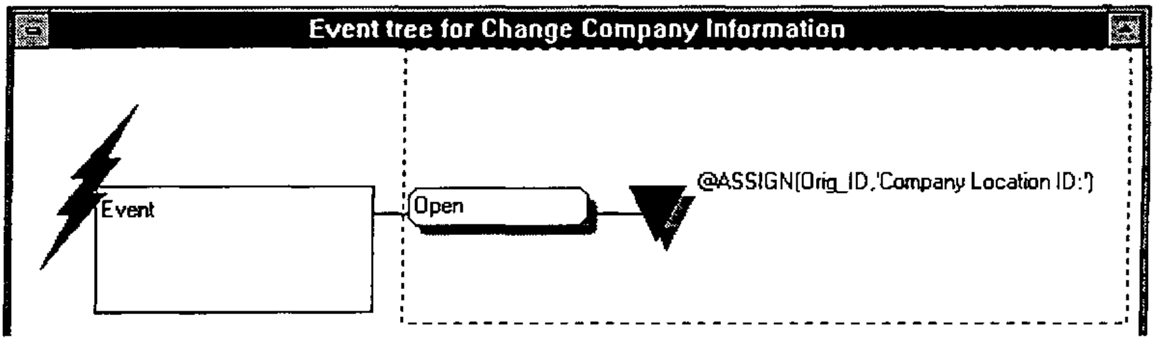
5. If "Filters..." is Clicked, the "Link Filters" Dialog is presented which allows the programmer to filter the contents of the data base before evaluation by the ObjectVision application

Visual "Source Code" for Data Link Creation (continued)



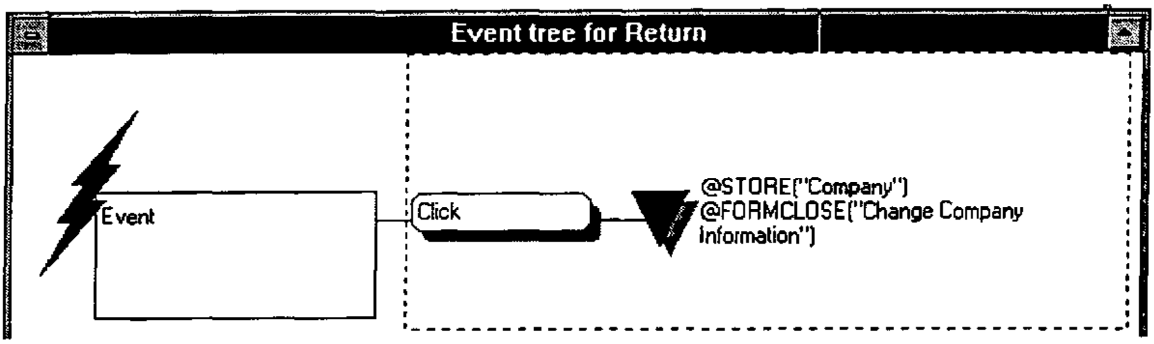
```
@IF(@ISBLANK('Company Location
ID:'),@EVENT("Edit Company Info","Click"),0)
```

Visual “Source Code” for a “Change Event” on the “Licensee ID” Field



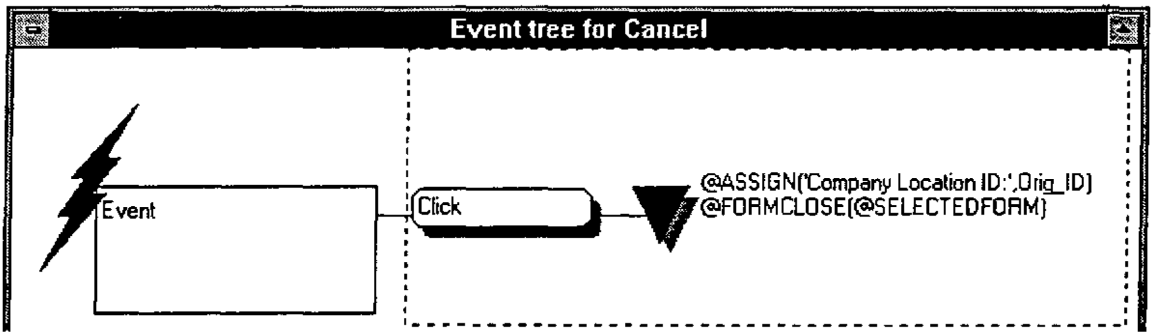
```
@ASSIGN(Orig_ID,'Company Location ID:')
```

Visual “Source Code” for an “Open Event” on the “Change Company Information” Form



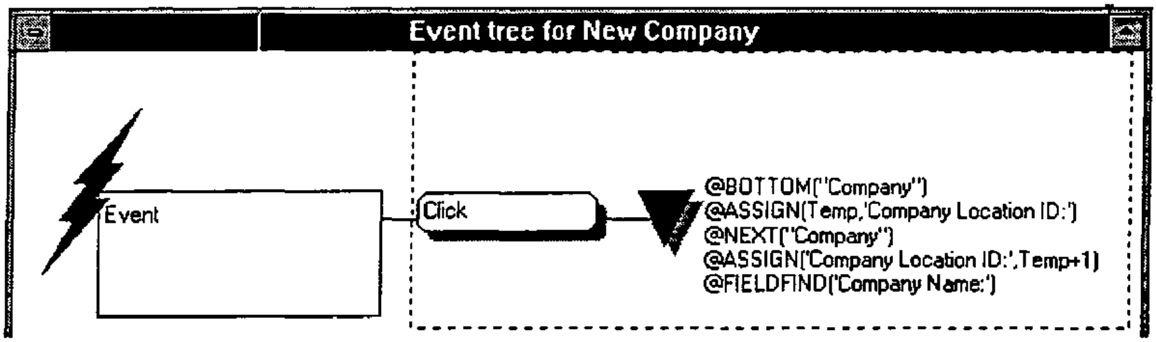
```
@STORE("Company")
@FORMCLOSE("Change Company
Information")
```

Visual “Source Code” for a “Click Event” on the “Return” Button on the “Change Company Information” Form



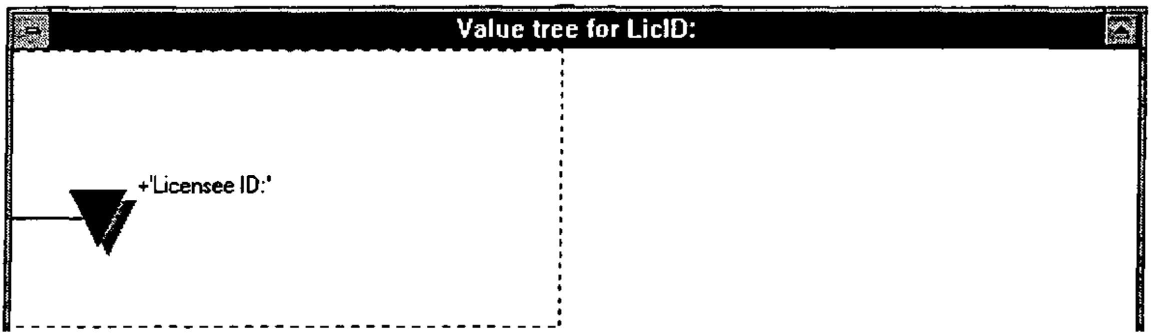
```
@ASSIGN('Company Location ID:',Orig_ID)
@FORMCLOSE(@SELECTEDFORM)
```

Visual “Source Code” for a “Click Event” on the “Cancel” Button on the “Change Company Information” Form



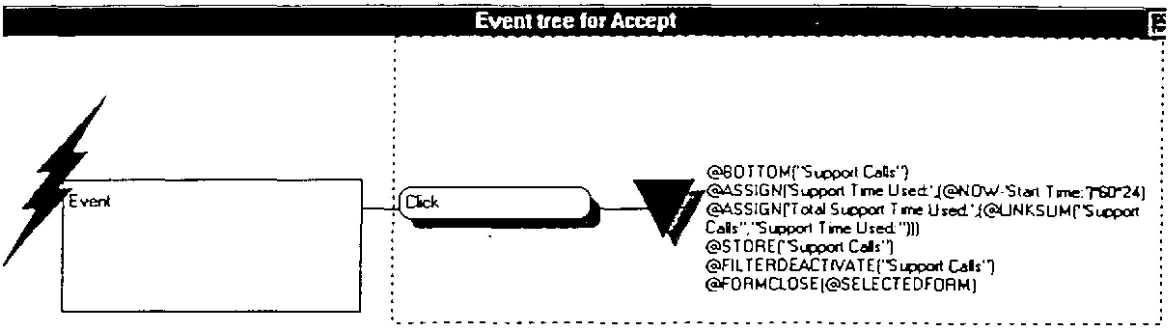
```
@BOTTOM("Company")
@ASSIGN(Temp,'Company Location ID:')
@NEXT("Company")
@ASSIGN('Company Location ID:',Temp+1)
@FIELDFIND('Company Name:')
```

Visual “Source Code” for a “Click Event” on the “New Company” Button on the “Change Company Information” Form



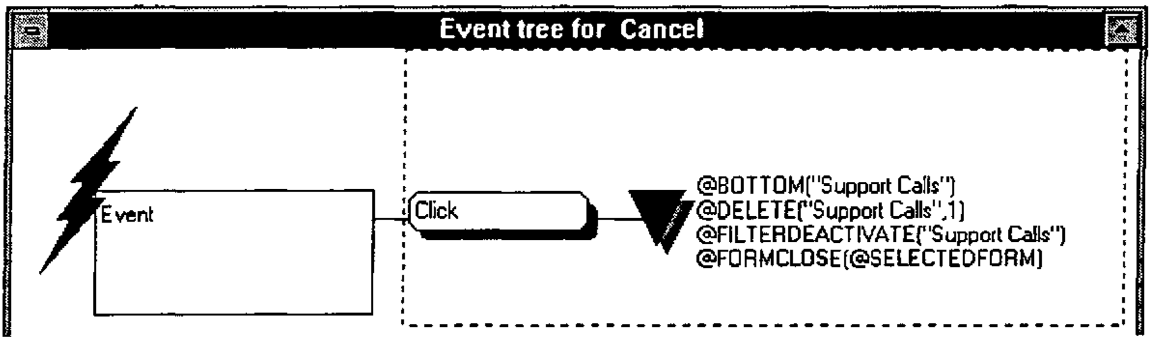
```
+'Licensee ID:'
```

Visual “Source Code” for assigning the value of the “LicID” Field on the “Support Call” Form



```
@BOTTOM("Support Calls")
@ASSIGN("Support Time Used:",(@NOW-"Start Time:")*60*24)
@ASSIGN("Total Support Time Used:",(@LINKSUM("Support
Calls","Support Time Used:")))
@STORE("Support Calls")
@FILTERDEACTIVATE("Support Calls")
@FORMCLOSE(@SELECTEDFORM)
```

Visual "Source Code" for a "Click Event" on the "Accept" Button
on the "Support Call" Form



```
@BOTTOM("Support Calls")
@DELETE("Support Calls",1)
@FILTERDEACTIVATE("Support Calls")
@FORMCLOSE(@SELECTEDFORM)
```

Visual "Source Code" for a "Click Event" on the "Cancel" Button
on the "Support Call" Form

APPENDIX D

Tic Tac Toe Design Package

Tic-Tac-Toe Requirements Definition Statement

The application shall provide a Graphic User Interface which allows a player to select Tic-Tac-Toe moves by clicking on a mouse-sensitive board and to begin the game by clicking on a <New game> button. The game shall respond by painting a blank Tic Tac Toe board and presenting a message to "click on a square or select <You Go First> to begin play." The game shall alternately accept a user's move and make its own move with the goal of winning the game. The system shall reject illegal moves attempted by the user and shall fill in (legal) moves made by the user and itself. The system shall monitor for a win or a draw and display an appropriate message. The player shall be "X" and the program shall be "O." No player records or statistics will be kept; each game shall be a clean start. The gaming strategy shall first rule out a win by the player (this should be impossible), then look for a win for itself, then look for a block of an imminent win by the player and then determine an offensive move.

Coad/Yourdon Object-Oriented Analysis

Classes/Objects:
(Domain Related)
Playing Board, with Tic Tac Toe icon
Cells (one for each play location)
Tokens ("X", "O")
Rows, Columns, Diagonals
Player
Strategies and Plays
(Program Related)
Window
Message Box (to communicate with User)
Controls (for starting a new game, quitting and letting the program go first)
Game Engine (to make moves on behalf of the application)

Gen-Spec Structure: None

Whole-Part Structure:
Window:Board|Controls|MessageBox
Board:Cells
Cells:Tokens
Board:Rows|Columns|Diagonals (R|C|D)

Attributes:
Cell.Value (internal integer representation of Token, -1 for "O", +1 for "X" and 0 for "blank")
Cell.Token (external string representation, including Font and Color)
Rows|Columns|Diagonals.Sum (an integer whose value is the sum of the three Cell.Value in that row, column or diagonal)
Window and Board Geometry (in general, such as color and border)

Services/Calculations:

On Cell
Monitor for Mouse-Click over Cell
Validate User Changes to Cell.Token (is Cell empty?)
Send a Message if Cell is taken (or game is over)
Set Cell.Token to "X" after valid User click on Cell
Set Cell.Value based on changes to Cell.Token
Deactivate the < You Go First> control (on first move)
Give control to Game Engine to make its move

Game Strategies
Look for User Win (any Sum = 3) (should be impossible)
Look for Game Winning Move (any Sum = -2)
Look for Blocking Move (any Sum = 2)
Look for a Wedge-prevention Move (to avoid the several ways a User might create a "double bind")
Pick a Cell according to the following search pattern:

2	6	3
7	1	8
4	9	5

On R|C|D Update the R|C|D.Sum whenever a member changes value

Application Navigate Cells and Controls when the User presses the <Tab> key
Emulate a Mouse-Click when the User presses other keys

On <New Game> control (when clicked)
Initialize all Cells to empty
Activate the <You Go First> control
Display a Message to the User

On <You Go First> control (when clicked)
The Game Engine will take the center Cell
Deactivate the < You Go First> control

On <Quit> control (when clicked)
Close the Application

Coad/Yourdon Object-Oriented Design

Note: For Visual Basic implementation, there is no inheritance and only Classes/Objects/Behaviors related to the User Interface

Human Interaction Component

User Classes: Tic Tac Toe Players (only one skill level; multiple skill levels is future scope)

Description: People who don't mind never being able to win a game they are playing

Command Hierarchy: New Game --> User First | Game First --> Alternating Moves
Quit Button Available at all times
Game Over when all Cells are Taken, or when User (impossible) or Game gets three of their Tokens in a row

Window: Titled "Tic Tac Toe"
Large enough to contain a Tic Tac Toe board, three buttons and a Message Box

Fields: Cell Array (9) Each consists of an editable TextBox (not sizable)
May contain a single <blank> (the default value), a large bold "X", or "O" (18point Sans Serif or equivalent)
The Mouse Pointer Icon should change when it is over the active area of a Cell
Each Cell should provide its own validation and updating services when clicked upon
If possible, the Cell should keep two values, one textual ("X","O", <blank>) and one numeric (+1, -1, 0)

Message Box A non-editable TextBox in which to display messages/prompts to the User
Sized to display up to 4, 40-character lines, with word wrap
Default contents should be "Click on <New Game> to begin."

Quit A Command Pushbutton which allows the User to exit the game
Caption reads "Quit"

New Game A Command Pushbutton which allows the User to start a new game
Caption reads "New Game"

You Go First A Command Pushbutton which allows the User to instruct the program to make the first move
Caption reads "You Go First"
The Command should only be visible and enabled just after <New Game> is clicked, but before the User has clicked on any Cell

Graphic Lines Four straight lines, organized to look like a traditional Tic Tac Toe board

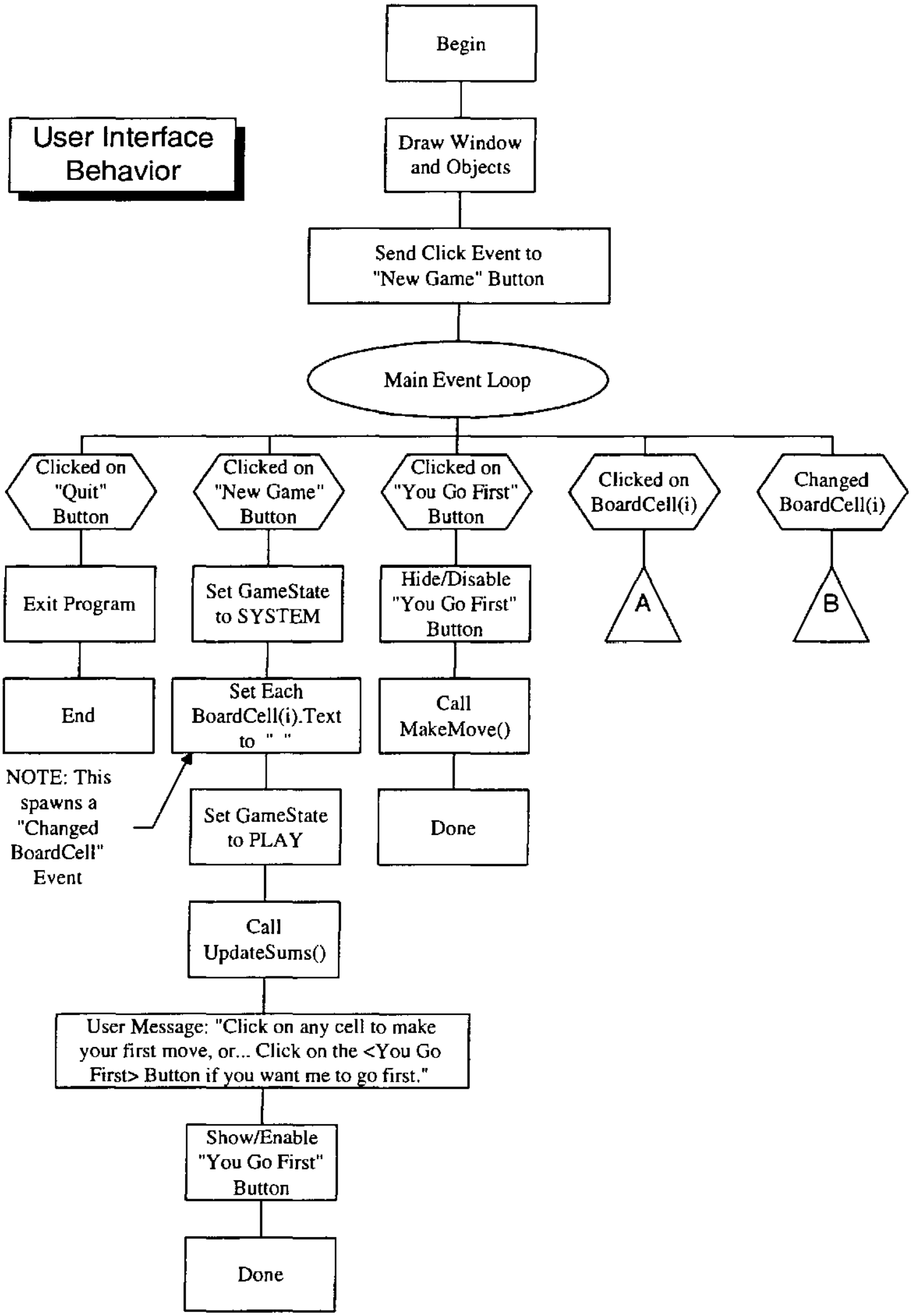
Note: The User should be able to operate the system without a mouse by using the <Tab> key to navigate the Board and Buttons, and any standard key to place an "X" or activate a button.

Note: Standard MSWindows pull-down menus (e.g., File, Edit, etc.) were deemed unnecessary for this application.

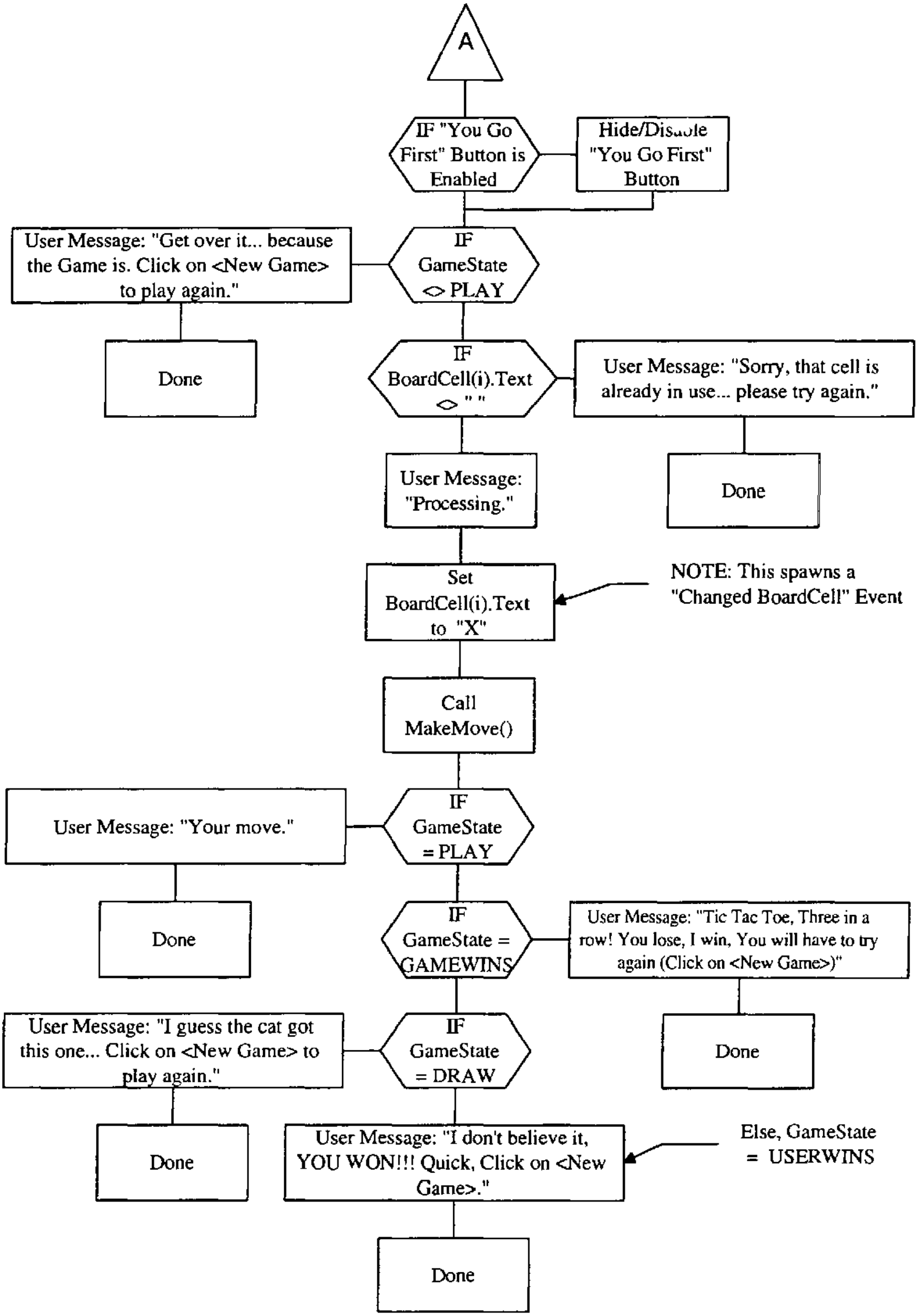
Task Management Component

Event Driven Tasks:	See User Interface and Game Engine Diagrams
Clock Driven Tasks	None
Priority/Critical Tasks	Not Applicable
Other Tasks	See User Interface and Game Engine Diagrams

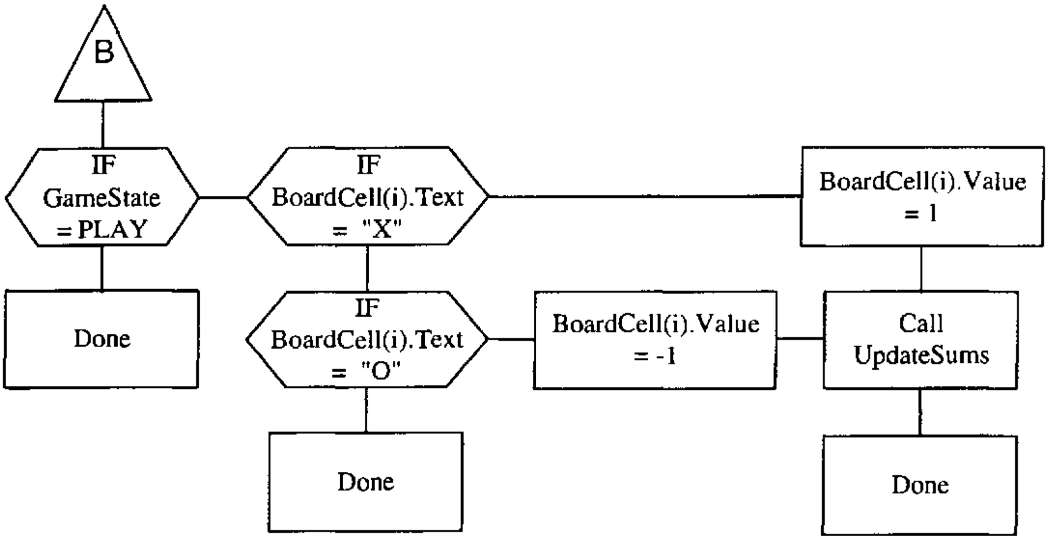
User Interface Service Diagram (Main Event Loop and Command Buttons)


User Interface Behavior
Begin
Draw Window and Objects
Send Click Event to "New Game" Button
Main Event Loop
Clicked on "Quit" Button
Clicked on "New Game" Button
Clicked on "You Go First" Button
Clicked on BoardCell(i)
Changed BoardCell(i)
Exit Program
Set GameState to SYSTEM
Hide/Disable "You Go First" Button
A
B
End
Set Each BoardCell(i).Text to " "
Call MakeMove()
NOTE: This spawns a "Changed BoardCell" Event
Set GameState to PLAY
Done
Call UpdateSums()
User Message: "Click on any cell to make your first move, or... Click on the <You Go First> Button if you want me to go first."
Show/Enable "You Go First" Button
Done

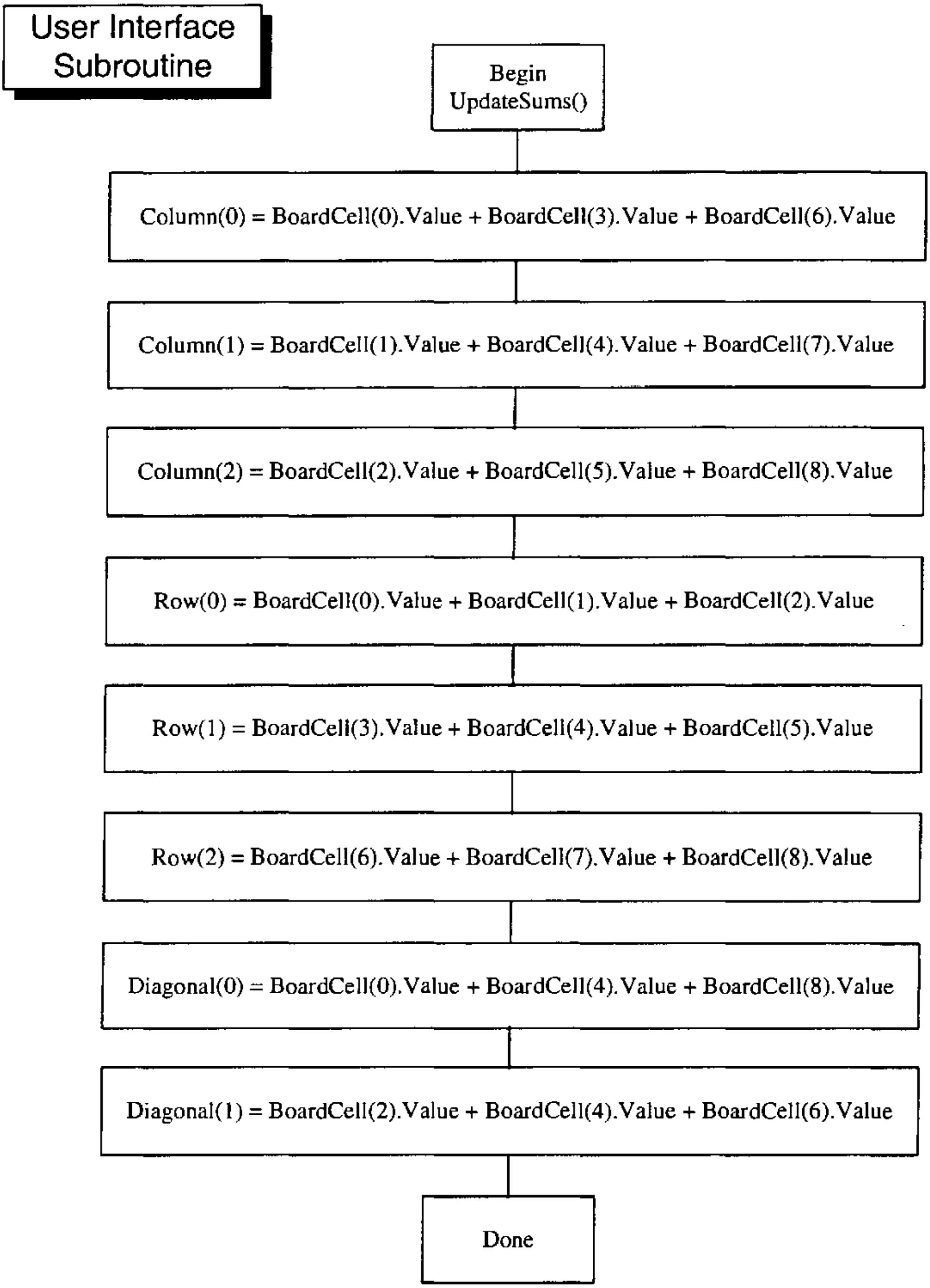
User Interface Service Diagram (Clicked on Text Cell)


A
IF "You Go First" Button is Enabled
Hide/Disable "You Go First" Button
User Message: "Get over it... because the Game is. Click on <New Game> to play again."
IF GameState <> PLAY
Done
IF BoardCell(i).Text <> " "
User Message: "Sorry, that cell is already in use... please try again."
User Message: "Processing."
Done
Set BoardCell(i).Text to "X"
NOTE: This spawns a "Changed BoardCell" Event
Call MakeMove()
User Message: "Your move."
IF GameState = PLAY
Done
IF GameState = GAMEWINS
User Message: "Tic Tac Toe, Three in a row! You lose, I win, You will have to try again (Click on <New Game>)"
User Message: "I guess the cat got this one... Click on <New Game> to play again."
IF GameState = DRAW
Done
Done
User Message: "I don't believe it, YOU WON!!! Quick, Click on <New Game>."
Else, GameState = USERWINS
Done

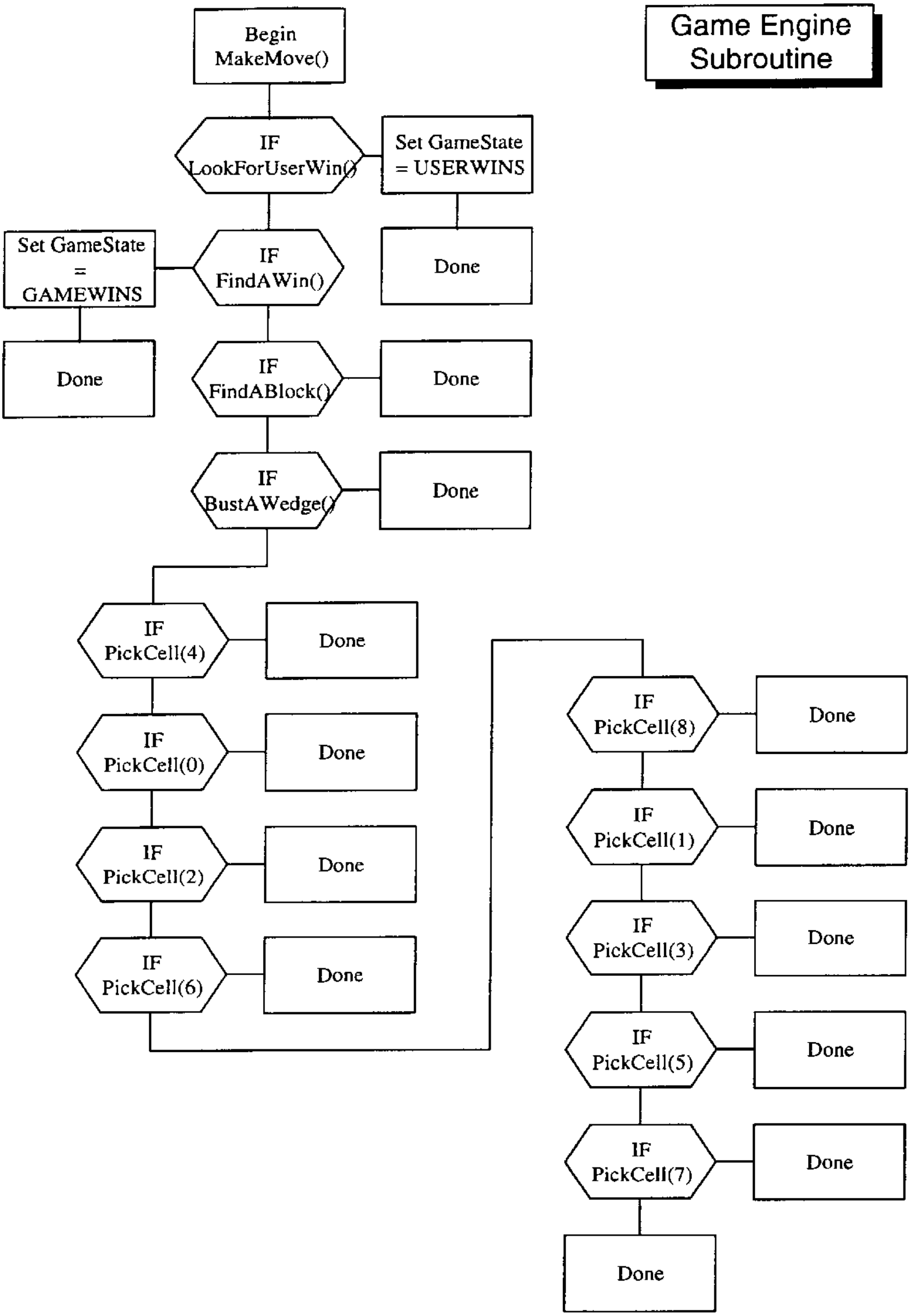
User Interface Service Diagram (Changed Value on Text Cell)



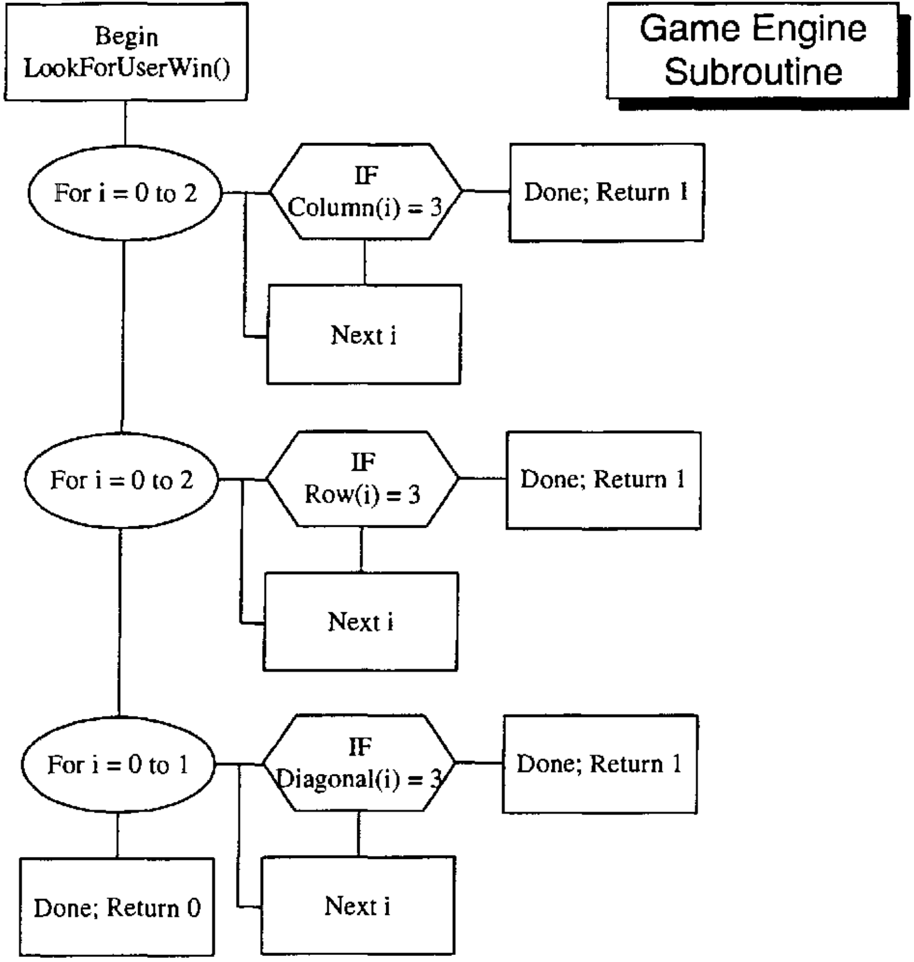
User Interface Service Diagram (Update Sums)

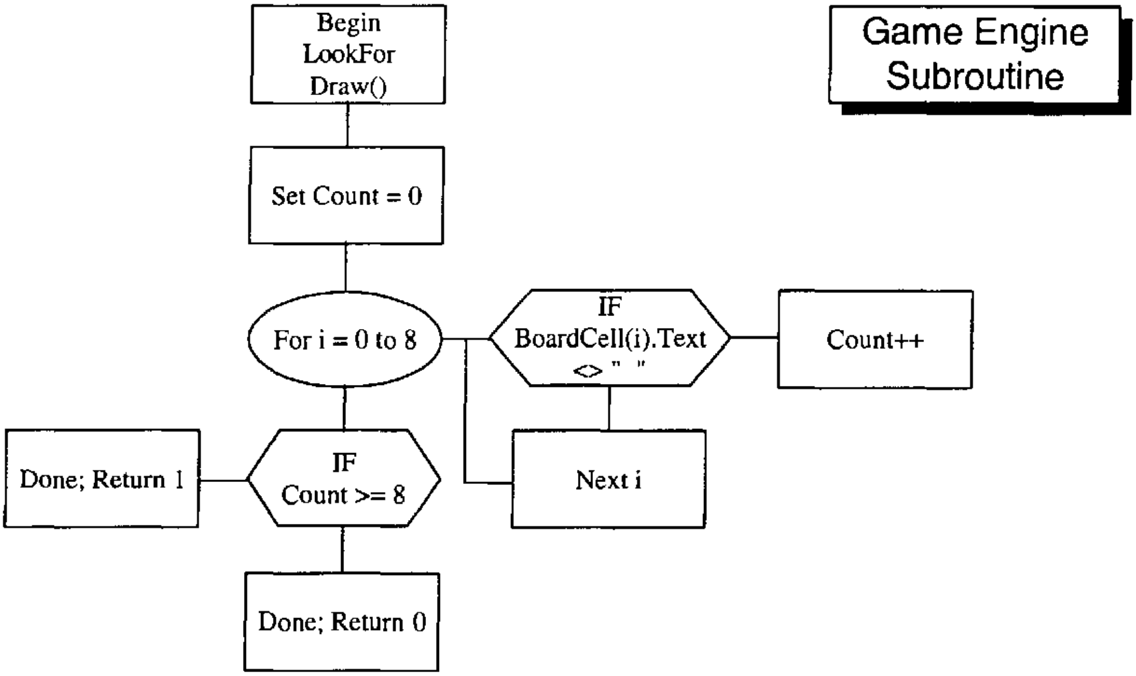

User Interface Subroutine
Begin UpdateSums()
Column(0) = BoardCell(0).Value + BoardCell(3).Value + BoardCell(6).Value
Column(1) = BoardCell(1).Value + BoardCell(4).Value + BoardCell(7).Value
Column(2) = BoardCell(2).Value + BoardCell(5).Value + BoardCell(8).Value
Row(0) = BoardCell(0).Value + BoardCell(1).Value + BoardCell(2).Value
Row(1) = BoardCell(3).Value + BoardCell(4).Value + BoardCell(5).Value
Row(2) = BoardCell(6).Value + BoardCell(7).Value + BoardCell(8).Value
Diagonal(0) = BoardCell(0).Value + BoardCell(4).Value + BoardCell(8).Value
Diagonal(1) = BoardCell(2).Value + BoardCell(4).Value + BoardCell(6).Value
Done

Game Engine Service Diagram (Main Move Selection Logic)

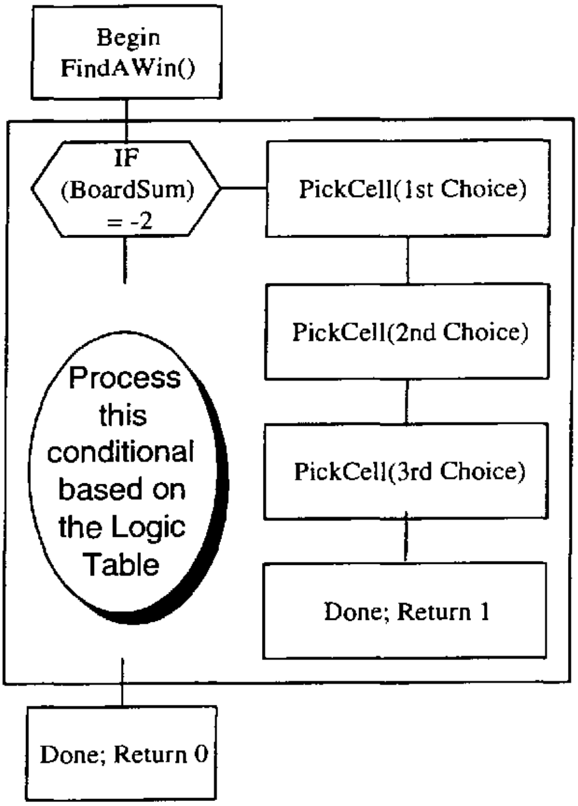


Game Engine Service Diagram (Look for User Win and Look for Draw)



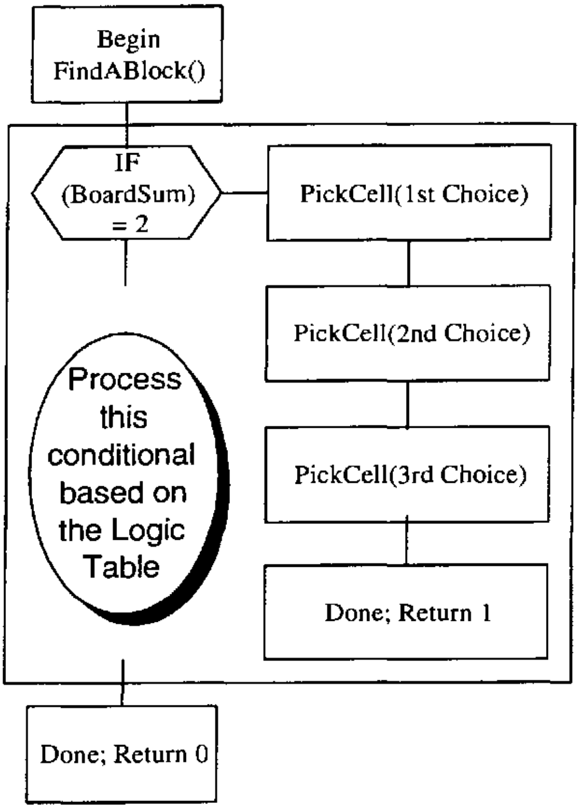


Game Engine Service Diagram (Look for Win and Look for Block)



Game Engine Subroutine

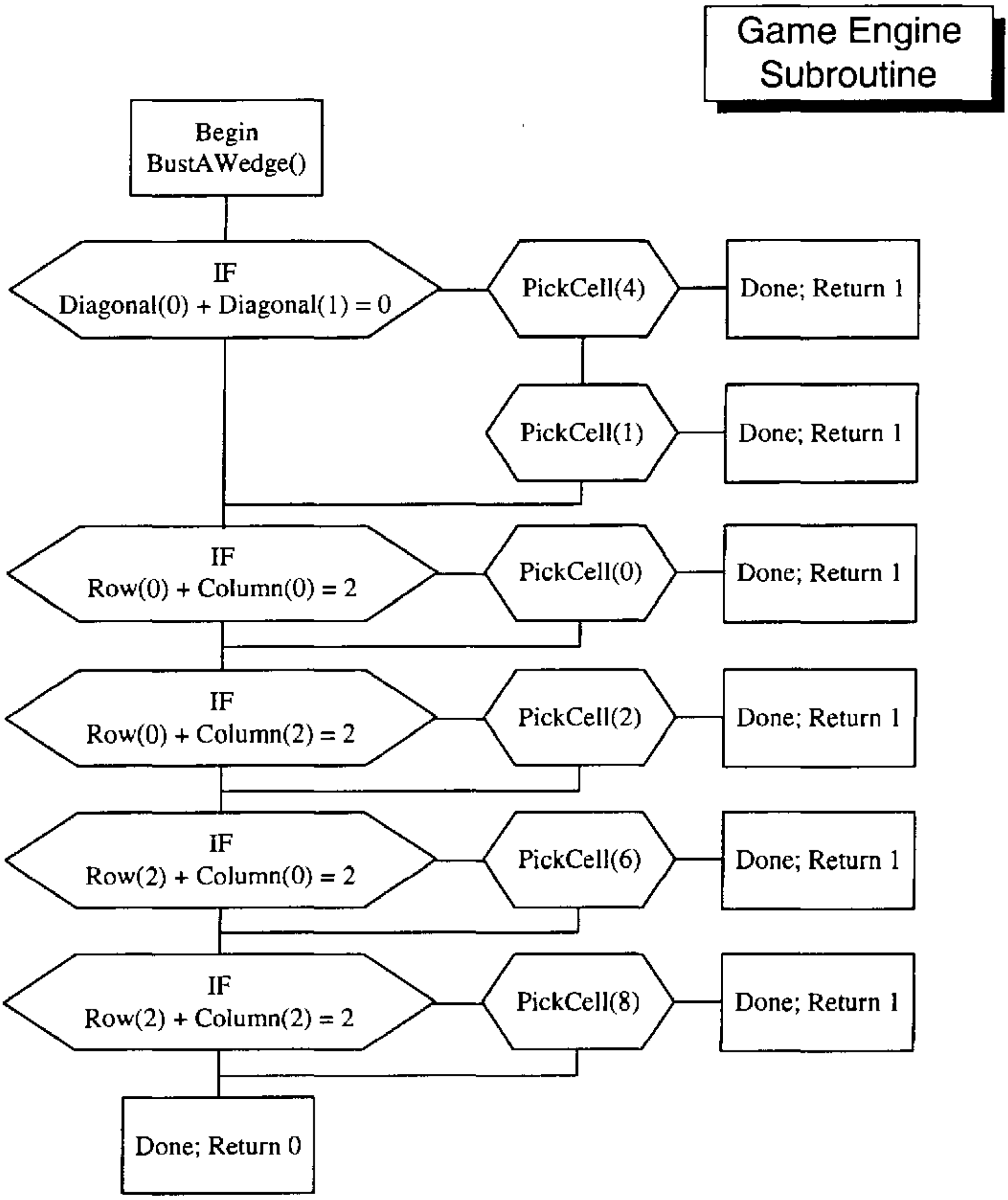
Logic Table			
BoardSum	1st Choice	2nd Choice	3rd Choice
Column(0)	0	3	6
Column(1)	1	4	7
Column(2)	2	5	8
Row(0)	0	1	2
Row(1)	3	4	5
Row(2)	6	7	8
Diagonal(0)	0	4	8
Diagonal(1)	2	4	6



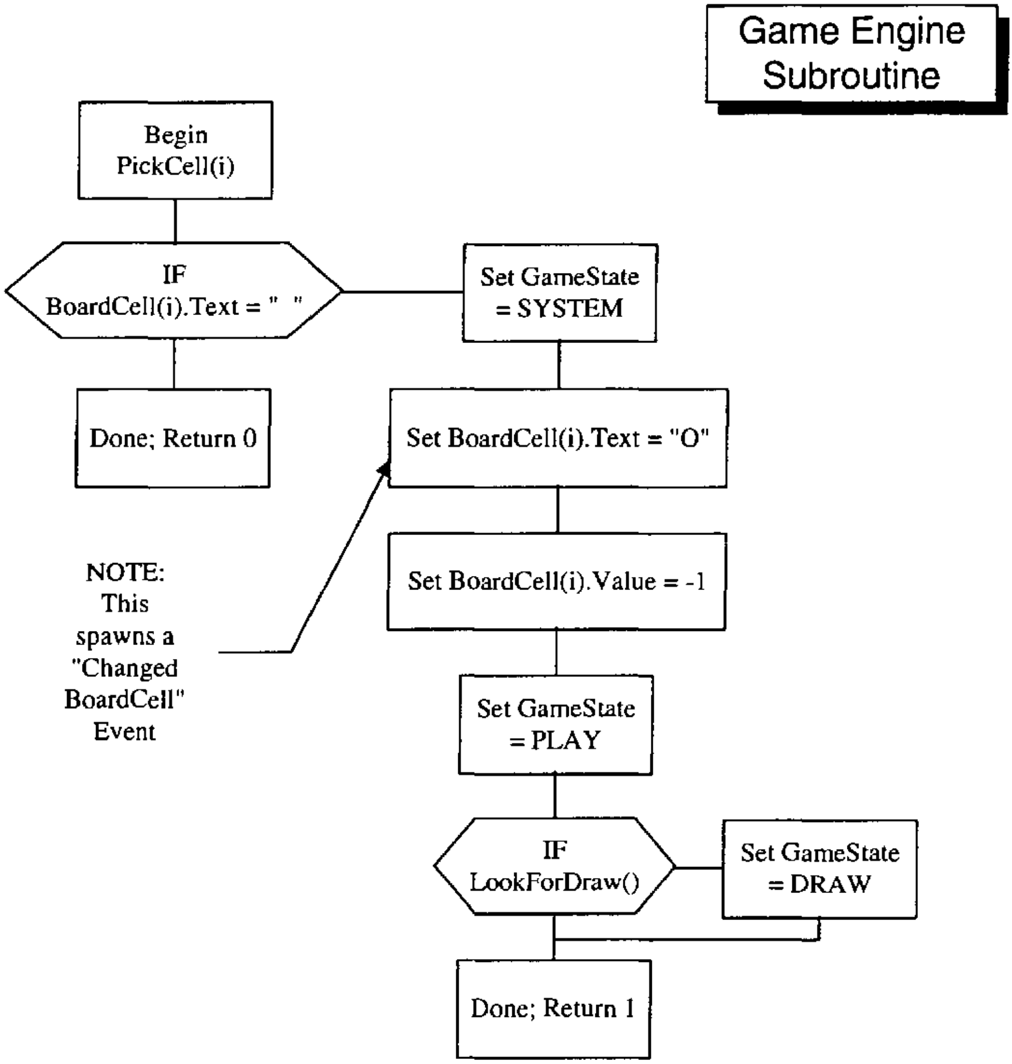
Game Engine Subroutine

Logic Table			
BoardSum	1st Choice	2nd Choice	3rd Choice
Column(0)	0	3	6
Column(1)	1	4	7
Column(2)	2	5	8
Row(0)	0	1	2
Row(1)	3	4	5
Row(2)	6	7	8
Diagonal(0)	0	4	8
Diagonal(1)	2	4	6

Game Engine Service Diagram (Bust a Wedge)

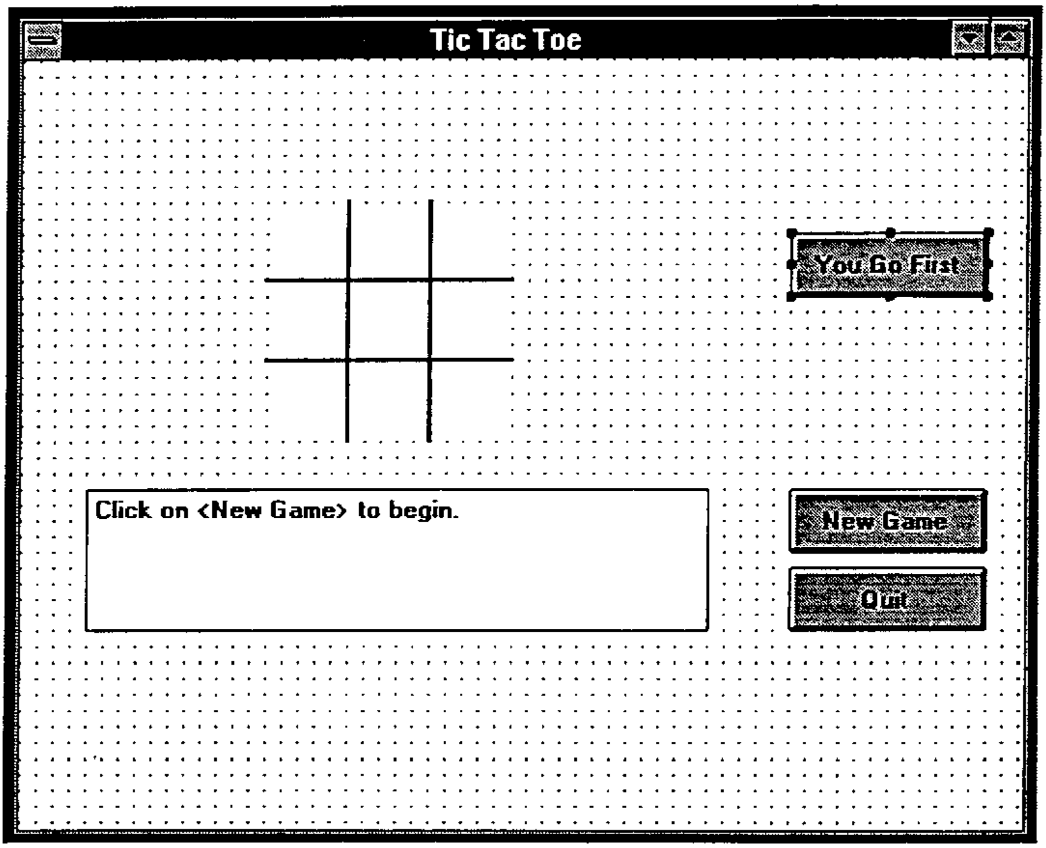


Game Engine Service Diagram (Pick a Cell)



APPENDIX E

Tic Tac Toe Visual Basic Listings/Screens



Click on <New Game> to begin.

Tic Tac Toe Main Screen (Visual Basic Version)

Main Form Object Definitions (MAIN.FRM)

```
VERSION 2.00
Begin Form frmMain
   BorderStyle     =   3  'Fixed Double
   Caption         =   "Tic Tac Toe"
   ClientHeight    =   5820
   ClientLeft      =   1065
   ClientTop       =   1740
   ClientWidth     =   7365
   Height          =   6225
   Left            =   1005
   LinkTopic       =   "Form1"
   ScaleHeight     =   5820
   ScaleWidth      =   7365
   Top             =   1395
   Width           =   7485
   Begin CommandButton btnQuit
      Caption         =   "Quit"
      Height          =   495
      Left            =   5640
      TabIndex        =   12
      Top             =   3840
      Width           =   1455
   End
   Begin TextBox txtCell
      Alignment       =   2  'Center
      BorderStyle     =   0  'None
      FontBold        =   -1  'True
      FontItalic      =   0   'False
      FontName        =   "MS Sans Serif"
      FontSize        =   18
      FontStrikethru  =   0   'False
      FontUnderline   =   0   'False
      Height          =   435
      Index           =   8
      Left            =   3120
      MousePointer    =   10  'Up Arrow
      TabIndex        =   11
      Text            =   " "
      Top             =   2400
      Width           =   375
   End
   Begin TextBox txtCell
      Alignment       =   2  'Center
      BorderStyle     =   0  'None
      FontBold        =   -1  'True
      FontItalic      =   0   'False
      FontName        =   "MS Sans Serif"
      FontSize        =   18
      FontStrikethru  =   0   'False
      FontUnderline   =   0   'False
      Height          =   435
```

```
  Index         = 7
  Left          = 2520
  MousePointer  = 10 'Up Arrow
  TabIndex      = 10
  Text          = " "
  Top           = 2400
  Width         = 375
End
Begin TextBox txtCell
  Alignment     = 2 'Center
  BorderStyle   = 0 'None
  FontBold      = -1 'True
  FontItalic    = 0 'False
  FontName      = "MS Sans Serif"
  FontSize      = 18
  FontStrikethru = 0 'False
  FontUnderline = 0 'False
  Height        = 435
  Index         = 6
  Left          = 1920
  MousePointer  = 10 'Up Arrow
  TabIndex      = 9
  Text          = " "
  Top           = 2400
  Width         = 375
End
Begin TextBox txtCell
  Alignment     = 2 'Center
  BorderStyle   = 0 'None
  FontBold      = -1 'True
  FontItalic    = 0 'False
  FontName      = "MS Sans Serif"
  FontSize      = 18
  FontStrikethru = 0 'False
  FontUnderline = 0 'False
  Height        = 435
  Index         = 5
  Left          = 3120
  MousePointer  = 10 'Up Arrow
  TabIndex      = 8
  Text          = " "
  Top           = 1800
  Width         = 375
End
Begin TextBox txtCell
  Alignment     = 2 'Center
  BorderStyle   = 0 'None
  FontBold      = -1 'True
  FontItalic    = 0 'False
  FontName      = "MS Sans Serif"
  FontSize      = 18
  FontStrikethru = 0 'False
  FontUnderline = 0 'False
```

```
   Height       = 435
   Index        = 4
   Left         = 2520
   MousePointer = 10 'Up Arrow
   TabIndex     = 7
   Text         = " "
   Top          = 1800
   Width        = 375
End
Begin TextBox txtCell
   Alignment       = 2  'Center
   BorderStyle     = 0  'None
   FontBold        = -1 'True
   FontItalic      = 0  'False
   FontName        = "MS Sans Serif"
   FontSize        = 18
   FontStrikethru  = 0  'False
   FontUnderline   = 0  'False
   Height          = 435
   Index           = 3
   Left            = 1920
   MousePointer    = 10 'Up Arrow
   TabIndex        = 6
   Text            = " "
   Top             = 1800
   Width           = 375
End
Begin TextBox txtCell
   Alignment       = 2  'Center
   BorderStyle     = 0  'None
   FontBold        = -1 'True
   FontItalic      = 0  'False
   FontName        = "MS Sans Serif"
   FontSize        = 18
   FontStrikethru  = 0  'False
   FontUnderline   = 0  'False
   Height          = 435
   Index           = 2
   Left            = 3120
   MousePointer    = 10 'Up Arrow
   TabIndex        = 5
   Text            = " "
   Top             = 1200
   Width           = 375
End
Begin TextBox txtCell
   Alignment       = 2  'Center
   BorderStyle     = 0  'None
   FontBold        = -1 'True
   FontItalic      = 0  'False
   FontName        = "MS Sans Serif"
   FontSize        = 18
   FontStrikethru  = 0  'False
```

```
      FontUnderline  =  0  'False
      Height      =  435
      Index       =  1
      Left       =  2520
      MousePointer  =  10 'Up Arrow
      TabIndex     =  4
      Text       =  " "
      Top        =  1200
      Width       =  375
   End
   Begin CommandButton btnYouGoFirst
      Caption      =  "You Go First"
      Enabled      =  0  'False
      Height      =  495
      Left       =  5640
      TabIndex     =  0
      Top        =  1320
      Visible      =  0  'False
      Width       =  1455
   End
   Begin CommandButton btnNewGame
      Caption      =  "New Game"
      Height      =  495
      Left       =  5640
      TabIndex     =  1
      Top        =  3240
      Width       =  1455
   End
   Begin TextBox txtCell
      Alignment     =  2  'Center
      BorderStyle    =  0  'None
      DragIcon     =  MAIN.FRX:0000
      FontBold     =  -1  'True
      FontItalic    =  0  'False
      FontName     =  "MS Sans Serif"
      FontSize     =  18
      FontStrikethru =  0  'False
      FontUnderline  =  0  'False
      Height      =  435
      Index       =  0
      Left       =  1920
      MousePointer  =  10 'Up Arrow
      TabIndex     =  3
      Text       =  " "
      Top        =  1200
      Width       =  375
   End
   Begin TextBox txtMsgBox
      Height      =  1095
      Left       =  480
      MultiLine     =  -1  'True
      TabIndex     =  2
      TabStop      =  0  'False
```

```
    Text         =  "Click on <New Game> to begin."
    Top          =  3240
    Width        =  4575
  End
  Begin Line Line2
    BorderWidth    =  2
    X1           =  3000
    X2           =  3000
    Y1           =  1080
    Y2           =  2880
  End
  Begin Label Label1
    Caption        =  "Label1"
    Height         =  375
    Index          =  0
    Left           =  3600
    TabIndex       =  13
    Top            =  1080
    Width          =  615
  End
  Begin Line Line4
    BorderWidth    =  2
    X1           =  1800
    X2           =  3600
    Y1           =  2280
    Y2           =  2280
  End
  Begin Line Line3
    BorderWidth    =  2
    X1           =  1800
    X2           =  3600
    Y1           =  1680
    Y2           =  1680
  End
  Begin Line Line1
    BorderWidth    =  2
    X1           =  2400
    X2           =  2400
    Y1           =  1080
    Y2           =  2880
  End
End
```

Main Form Object Behaviors (MAIN.FRM)

```
Sub btnNewGame_Click ()

   Dim i As Integer

'Blank out board
   gnGameState = SYSTEM_CONTROL
   For i = 0 To 8
      txtCell(i).Tag = 0
      txtCell(i).Text = " "
      Label1(i).Caption = txtCell(i).Tag
   Next

'Set up for play
   gnGameState = PLAY

   Call UpdateSums

   txtMsgBox.Text = "Click on any cell to make your first move, or... Click on the <You Go First>
          button if you want me to go first."

   btnYouGoFirst.Enabled = True
   btnYouGoFirst.Visible = True

End Sub

Sub btnQuit_Click ()

   End

End Sub

Sub btnYouGoFirst_Click ()

Dim temp As Integer

   btnYouGoFirst.Enabled = False
   btnYouGoFirst.Visible = False

   temp = PickCell(4)

End Sub

Sub Form_Load ()
   Call btnNewGame_Click
End Sub

Sub txtCell_Change (Index As Integer)

   If gnGameState = PLAY Then
      If Val(txtCell(Index).Tag) = 1 Then
         txtCell(Index).Text = "X"
```

```
          ElseIf Val(txtCell(Index).Tag) = -1 Then
             txtCell(Index).Text = "O"
             Else txtCell(Index).Text = "X"
                  txtCell(Index).Tag = 1
                  Label1(Index).Caption = txtCell(Index).Tag
                'Update Col|Row|Diag Sums
                   Call UpdateSums
       End If
    End If

 End Sub

 Sub txtCell_Click (Index As Integer)

 'Just in case the User is faster than the system
    If gnGameState = SYSTEM_CONTROL Then
       Exit Sub
    End If

 'Get rid of <You Go First> if it is still there
    If btnYouGoFirst.Enabled = True Then
       btnYouGoFirst.Enabled = False
       btnYouGoFirst.Visible = False
    End If

 'See if game is still in progress
    If gnGameState <> PLAY Then
       txtMsgBox.Text = "Get over it... because the game is.  Click on <New Game> to play again."
       Exit Sub
    End If

 'Validate User Move
    If txtCell(Index).Text <> " " Then
       txtMsgBox.Text = "Sorry, that cell is already in use...please try again."
       Exit Sub
    End If

 'Set Cell to X
    txtMsgBox.Text = "Processing..." 'this invokes txtCell_Change()
    txtCell(Index).Text = "X"

 'Let program make its move
    Call MakeMove

 'Handle Game State

    Select Case (gnGameState)
       Case PLAY
          txtMsgBox.Text = "Your Move"
       Case GAME_WINS
          txtMsgBox.Text = "Tic Tac Toe, Three in a Row... You lose, I win! You will have to try again
            (Click on <New Game>)"
       Case DRAW
```

```
        txtMsgBox.Text = "I guess the cat got this one...Click on <New Game> to play again."
      Case USER_WINS
        txtMsgBox.Text = "I don't believe it, YOU WON!!!...Quick, click on <New Game>."
    End Select

End Sub

Sub txtCell_KeyPress (Index As Integer, KeyAscii As Integer)
   Call txtCell_Click(Index)
End Sub

Global varColTot(3) As Integer
Global varRowTot(3) As Integer
Global varDiagTot(2) As Integer
Global gnGameState As Integer

Global Const SYSTEM_CONTROL = -1
Global Const PLAY = 0
Global Const GAME_WINS = 1
Global Const DRAW = 2
Global Const USER_WINS = 3
```

Supporting Functions and Subroutines (TTT.BAS)

```
Function BustAWedge ()

  If (varDiagTot(0) + varDiagTot(1)) = 0 Then
    If PickCell(4) Then
      BustAWedge = 1
      Exit Function
    End If
    If PickCell(1) Then
      BustAWedge = 1
      Exit Function
    End If
  End If

  If (varRowTot(0) + varColTot(0)) = 2 Then
    If PickCell(0) Then
      BustAWedge = 1
      Exit Function
    End If
  End If

  If (varRowTot(0) + varColTot(2)) = 2 Then
    If PickCell(2) Then
      BustAWedge = 1
      Exit Function
    End If
  End If

  If (varRowTot(2) + varColTot(0)) = 2 Then
    If PickCell(6) Then
      BustAWedge = 1
      Exit Function
    End If
  End If

  If (varRowTot(2) + varColTot(2)) = 2 Then
    If PickCell(8) Then
      BustAWedge = 1
      Exit Function
    End If
  End If

  BustAWedge = 0

End Function
```

```
Function FindABlock ()

   Dim i As Integer
   If varColTot(0) = 2 Then

      i = PickCell(0)
      i = PickCell(3)
      i = PickCell(6)
      FindABlock = 1
      Exit Function
   End If


   If varColTot(1) = 2 Then
      i = PickCell(1)
      i = PickCell(4)
      i = PickCell(7)
      FindABlock = 1
      Exit Function
   End If

   If varColTot(2) = 2 Then
      i = PickCell(2)
      i = PickCell(5)
      i = PickCell(8)
      FindABlock = 1
      Exit Function
   End If

   If varRowTot(0) = 2 Then
      i = PickCell(0)
      i = PickCell(1)
      i = PickCell(2)
      FindABlock = 1
      Exit Function
   End If

   If varRowTot(1) = 2 Then
      i = PickCell(3)
      i = PickCell(4)
      i = PickCell(5)
      FindABlock = 1
      Exit Function
   End If

   If varRowTot(2) = 2 Then
      i = PickCell(6)
      i = PickCell(7)
      i = PickCell(8)
      FindABlock = 1
      Exit Function
   End If
```

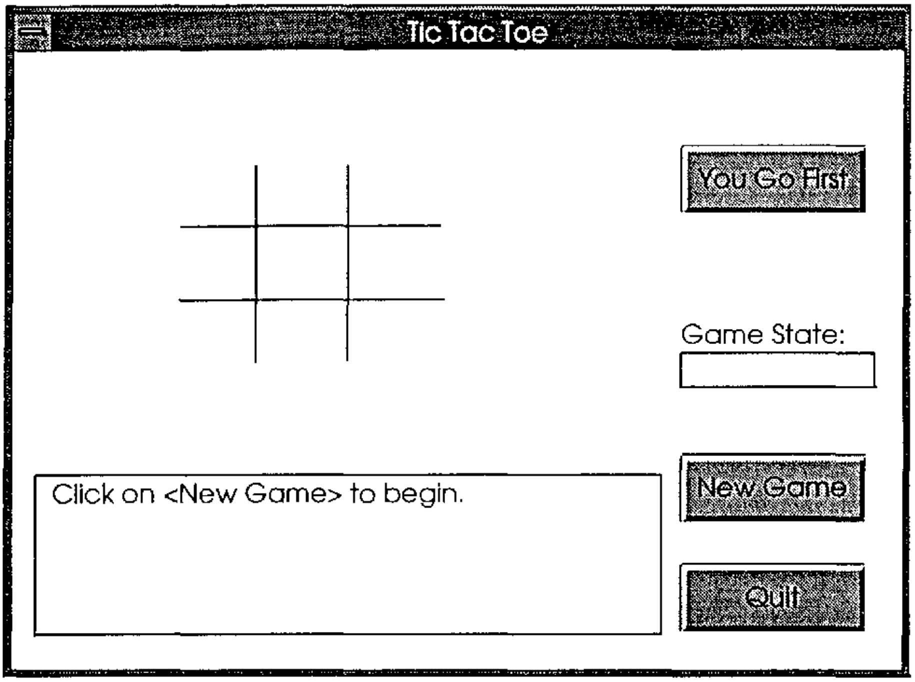
```
    If varDiagTot(0) = 2 Then
        i = PickCell(0)
        i = PickCell(4)
        i = PickCell(8)
        FindABlock = 1
        Exit Function
    End If

    If varDiagTot(1) = 2 Then
        i = PickCell(2)
        i = PickCell(4)
        i = PickCell(6)
        FindABlock = 1
        Exit Function
    End If

End Function

Function FindAWin ()
    Dim i As Integer
    If varColTot(0) = -2 Then

        i = PickCell(0)
        i = PickCell(3)
        i = PickCell(6)
        FindAWin = 1
        Exit Function
    End If


    If varColTot(1) = -2 Then
        i = PickCell(1)
        i = PickCell(4)
        i = PickCell(7)
        FindAWin = 1
        Exit Function
    End If

    If varColTot(2) = -2 Then
        i = PickCell(2)
        i = PickCell(5)
        i = PickCell(8)
        FindAWin = 1
        Exit Function
    End If

    If varRowTot(0) = -2 Then
        i = PickCell(0)
        i = PickCell(1)
        i = PickCell(2)
        FindAWin = 1
        Exit Function
    End If

    If varRowTot(1) = -2 Then
        i = PickCell(3)
        i = PickCell(4)
        i = PickCell(5)
        FindAWin = 1
        Exit Function
    End If

    If varRowTot(2) = -2 Then
        i = PickCell(6)
        i = PickCell(7)
        i = PickCell(8)
        FindAWin = 1
        Exit Function
    End If

    If varDiagTot(0) = -2 Then
        i = PickCell(0)
        i = PickCell(4)
        i = PickCell(8)
        FindAWin = 1
        Exit Function
    End If

    If varDiagTot(1) = -2 Then
        i = PickCell(2)
        i = PickCell(4)
        i = PickCell(6)
        FindAWin = 1
        Exit Function
    End If

End Function

Function LookForDraw ()

    Dim count, i As Integer

    count = 0

    For i = 0 To 8
        If frmMain.txtCell(i).Text <> " " Then
            count = count + 1
        End If
    Next

    If count >= 8 Then
        LookForDraw = 1
        Exit Function
    End If
```

```
  LookForDraw = 0

End Function

Function LookForUserWin ()

Dim i As Integer

  For i = 0 To 2
    If varColTot(i) = 3 Then
      LookForUserWin = 1
      Exit Function
    End If
  Next

  For i = 0 To 2
    If varRowTot(i) = 3 Then
      LookForUserWin = 1
      Exit Function
    End If
  Next

  For i = 0 To 1
    If varDiagTot(i) = 3 Then
      LookForUserWin = 1
      Exit Function
    End If
  Next

  LookForUserWin = 0

End Function

Sub MakeMove ()
  Dim i

  If LookForUserWin() Then
    Beep
    gnGameState = USER_WINS
    Exit Sub
  End If

  If FindAWin() Then
    Beep
    gnGameState = GAME_WINS
    Exit Sub
  End If

  If FindABlock() Then
    Exit Sub
  End If

  If BustAWedge() Then
```

```
        Exit Sub
    End If


    If PickCell(4) Then
        i = 4
        Exit Sub
    End If
    If PickCell(0) Then
        i = 0
        Exit Sub
    End If
    If PickCell(2) Then
        i = 2
        Exit Sub
    End If
    If PickCell(6) Then
        i = 6
        Exit Sub
    End If
    If PickCell(8) Then
        i = 8
        Exit Sub
    End If
    If PickCell(1) Then
        i = 1
        Exit Sub
    End If
    If PickCell(3) Then
        i = 3
        Exit Sub
    End If
    If PickCell(5) Then
        i = 5
        Exit Sub
    End If
    If PickCell(7) Then
        i = 7
        Exit Sub
    End If

End Sub

Function PickCell (Index As Integer)

    If frmMain.txtCell(Index).Text = " " Then
        gnGameState = SYSTEM_CONTROL
        frmMain.txtCell(Index).Text = "O"
        frmMain.txtCell(Index).Tag = -1
        frmMain.Label1(Index).Caption = frmMain.txtCell(Index).Tag
        gnGameState = PLAY

        If LookForDraw() Then
        Beep
        gnGameState = DRAW
      End If

      PickCell = 1
      Exit Function
    End If

    PickCell = 0
    Exit Function

  End Function

  Sub UpdateSums ()

    varColTot(0) = Val(frmMain.txtCell(0).Tag) + Val(frmMain.txtCell(3).Tag) +
          Val(frmMain.txtCell(6).Tag)
    varColTot(1) = Val(frmMain.txtCell(1).Tag) + Val(frmMain.txtCell(4).Tag) +
          Val(frmMain.txtCell(7).Tag)
    varColTot(2) = Val(frmMain.txtCell(2).Tag) + Val(frmMain.txtCell(5).Tag) +
          Val(frmMain.txtCell(8).Tag)

    varRowTot(0) = Val(frmMain.txtCell(0).Tag) + Val(frmMain.txtCell(1).Tag) +
          Val(frmMain.txtCell(2).Tag)
    varRowTot(1) = Val(frmMain.txtCell(3).Tag) + Val(frmMain.txtCell(4).Tag) +
          Val(frmMain.txtCell(5).Tag)
    varRowTot(2) = Val(frmMain.txtCell(6).Tag) + Val(frmMain.txtCell(7).Tag) +
          Val(frmMain.txtCell(8).Tag)

    varDiagTot(0) = Val(frmMain.txtCell(0).Tag) + Val(frmMain.txtCell(4).Tag) +
          Val(frmMain.txtCell(8).Tag)
    varDiagTot(1) = Val(frmMain.txtCell(2).Tag) + Val(frmMain.txtCell(4).Tag) +
          Val(frmMain.txtCell(6).Tag)

  End Sub
```

APPENDIX F

Tic Tac Toe Smart Elements Listings/Screens



Click on <New Game> to begin.

Tic Tac Toe Main Screen (Smart Elements Version)

Open Interface Resource File for Tic Tac Toe (TTT_SE.RC)

Note: For better readability, the object scripts were pulled from the individual objects and put into the "Script File" which follows this Resource File listing. Also, objects and operations not related to the application (i.e., overhead) were removed.

```
(Win.Compile
        Name:   "ttt_se.winMain"
        Version: 8
        Flags:   0x0001
        Deco:    0x0007
        MinWidth:       100
        MinHeight:      40
        LabelColor:     "Win.DefLabelColor"
        FocusColor:     "Win.DefFocusColor"
        IconFont:       "Win.DefIconFont"
        Icon:   "Win.DefIcon"
        PosFlags:       0x0001
        OptFlags:       0x0010
        DpiX:   78
        DpiY:   78
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        Label:  "Tic Tac Toe"
        FgColor:        "Win.DefFgColor"
        BgColor:        "Win.DefBgColor"
        Font:   "Win.DefFont"
        Pen:    "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.DefArrow"
        X:      125
        Y:      50
        W:      440
        H:      310
        WgtFlags:       0x0001
        Script: <MOVED TO SCRIPT FILE>
)

(PBut.Compile
        Name:   "ttt_se.winMain.btnQuit"
        Version: 8
        Label:  "Quit"
        FgColor:        "TBut.DefFgColor"
        BgColor:        "TBut.DefBgColor"
        Font:   "TBut.DefFont"
        Pen:    "Wgt.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.DefArrow"
        X:      331
        Y:      257
        W:      91
        H:      34
        WgtFlags:       0x0001
        Script: "on event TBUT_HIT\n\tWIN_Terminate(WGT_GetWin(SELF));\nend event\n"
)

(PBut.Compile
        Name:   "ttt_se.winMain.btnNewGame"
        Version: 8
        Label:  "New Game"
        FgColor:        "TBut.DefFgColor"
        BgColor:        "TBut.DefBgColor"
        Font:   "TBut.DefFont"
        Pen:    "Wgt.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.DefArrow"
        Index:  1
        X:      331
        Y:      202
        W:      91
        H:      34
        WgtFlags:       0x0001
        Script: <MOVED TO SCRIPT FILE>
)

(PBut.Compile
        Name:   "ttt_se.winMain.btnYouGoFirst"
        Version: 8
        Label:  "You Go First"
        FgColor:        "TBut.DefFgColor"
        BgColor:        "TBut.DefBgColor"
        Font:   "TBut.DefFont"
        Pen:    "Wgt.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.DefArrow"
        Index:  2
        X:      331
        Y:      47
        W:      91
        H:      34
        WgtFlags:       0x0001
        Script: <MOVED TO SCRIPT FILE>
)

(Panel.Compile
        Name:   "ttt_se.winMain.pnlBoard"
        Version: 8
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        LabelJustif:    0x0001
        FgColor:        "Panel.DefFgColor"
        BgColor:        "Panel.DefBgColor"
        Font:    "Wgt.DefFont"
        Pen:     "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor:  "Curs.DefArrow"
        Index:   3
        X:       81
        Y:       42
        W:       136
        H:       136
        WgtFlags:       0x0001
)

(IArea.Compile
        Name:    "ttt_se.winMain.pnlBoard.imgBoard"
        Version: 8
        Icon:    "IArea.DefIcon"
        FgColor:        "IArea.DefFgColor"
        BgColor:        "IArea.DefBgColor"
        Font:    "Wgt.DefFont"
        Pen:     "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor:  "Curs.DefArrow"
        W:       311
        H:       186
        WgtFlags:       0x0001
)

(LBox.Compile
        Name:    "ttt_se.winMain.pnlBoard.lbCell1"
        Version: 8
        StartCol: 0x0001
        StartRow: 0x0001
        ColWidth: 0x0032
        RowHeight: 0x0014
        ColNum:         1
        RowNum:         1
        LbKeys: "LBox.KeysDef"
        CellPen: "Win.DefPen"
        TextEditor:     "NMsgEd.EditTEd"
        SbSepW:         4
        SbSepH: 4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        FgColor:        "LBox.DefFgColor"
        BgColor:        "LBox.DefBgColor"
        Font:    "ttt_se.Font2"
        Pen:     "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor:  "Curs.Cross"
        Index:   1
        X:       5
        Y:      13
        W:      26
        H:      31
        WgtFlags:       0x0001
        Script: <MOVED TO SCRIPT FILE>
)

(LBox.Compile
        Name:   "ttt_se.winMain.pnlBoard.lbCell2"
        Version: 8
        StartCol: 0x0001
        StartRow: 0x0001
        ColWidth: 0x0032
        RowHeight: 0x0014
        ColNum:         1
        RowNum:         1
        LbKeys: "LBox.KeysDef"
        CellPen: "Win.DefPen"
        TextEditor:     "NMsgEd.EditTEd"
        SbSepW:         4
        SbSepH: 4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        FgColor:        "LBox.DefFgColor"
        BgColor:        "LBox.DefBgColor"
        Font:   "ttt_se.Font2"
        Pen:    "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.Cross"
        Index:  2
        X:      50
        Y:      13
        W:      26
        H:      31
        WgtFlags:       0x0001
        Script: <MOVED TO SCRIPT FILE>
)

(LBox.Compile
        Name:   "ttt_se.winMain.pnlBoard.lbCell3"
        Version: 8
        StartCol: 0x0001
        StartRow: 0x0001
        ColWidth: 0x0032
        RowHeight: 0x0014
        ColNum:         1
        RowNum:         1
        LbKeys: "LBox.KeysDef"
        CellPen: "Win.DefPen"
        TextEditor:     "NMsgEd.EditTEd"
        SbSepW:         4
        SbSepH: 4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        FgColor:        "LBox.DefFgColor"
        BgColor:        "LBox.DefBgColor"
        Font:     "ttt_se.Font2"
        Pen:      "Win.DefPen"
        Pattern:  "Patt.Empty"
        Cursor:   "Curs.Cross"
        Index:    3
        X:        95
        Y:        13
        W:        26
        H:        31
        WgtFlags:       0x0001
        Script:   <MOVED TO SCRIPT FILE>
)

(LBox.Compile
        Name:     "ttt_se.winMain.pnlBoard.lbCell4"
        Version: 8
        StartCol: 0x0001
        StartRow: 0x0001
        ColWidth: 0x0032
        RowHeight: 0x0014
        ColNum:         1
        RowNum:         1
        LbKeys: "LBox.KeysDef"
        CellPen: "Win.DefPen"
        TextEditor:     "NMsgEd.EditTEd"
        SbSepW:         4
        SbSepH: 4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        FgColor:        "LBox.DefFgColor"
        BgColor:        "LBox.DefBgColor"
        Font:     "ttt_se.Font2"
        Pen:      "Win.DefPen"
        Pattern:  "Patt.Empty"
        Cursor:   "Curs.Cross"
        Index:    4
        X:        5
        Y:        48
        W:        26
        H:        31
        WgtFlags:       0x0001
        Script:   <MOVED TO SCRIPT FILE>
)
```

```
(LBox.Compile
        Name:   "ttt_se.winMain.pnlBoard.lbCell5"
        Version: 8
        StartCol: 0x0001
        StartRow: 0x0001
        ColWidth: 0x0032
        RowHeight: 0x0014
        ColNum:         1
        RowNum:         1
        LbKeys: "LBox.KeysDef"
        CellPen: "Win.DefPen"
        TextEditor:     "NMsgEd.EditTEd"
        SbSepW:         4
        SbSepH: 4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        FgColor:        "LBox.DefFgColor"
        BgColor:        "LBox.DefBgColor"
        Font:   "ttt_se.Font2"
        Pen:    "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.Cross"
        Index:  5
        X:      50
        Y:      48
        W:      26
        H:      31
        WgtFlags:       0x0001
        Script: <MOVED TO SCRIPT FILE>
)

(LBox.Compile
        Name:   "ttt_se.winMain.pnlBoard.lbCell6"
        Version: 8
        StartCol: 0x0001
        StartRow: 0x0001
        ColWidth: 0x0032
        RowHeight: 0x0014
        ColNum:         1
        RowNum:         1
        LbKeys: "LBox.KeysDef"
        CellPen: "Win.DefPen"
        TextEditor:     "NMsgEd.EditTEd"
        SbSepW:         4
        SbSepH: 4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        FgColor:        "LBox.DefFgColor"
        BgColor:        "LBox.DefBgColor"
        Font:    "ttt_se.Font2"
        Pen:     "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor:  "Curs.Cross"
        Index:   6
        X:       95
        Y:       48
        W:       26
        H:       31
        WgtFlags:        0x0001
        Script:  <MOVED TO SCRIPT FILE>
)

(LBox.Compile
        Name:    "ttt_se.winMain.pnlBoard.lbCell7"
        Version: 8
        StartCol: 0x0001
        StartRow: 0x0001
        ColWidth: 0x0032
        RowHeight: 0x0014
        ColNum:          1
        RowNum:          1
        LbKeys: "LBox.KeysDef"
        CellPen: "Win.DefPen"
        TextEditor:      "NMsgEd.EditTEd"
        SbSepW:          4
        SbSepH: 4
        KeysNext:        "Panel.KeysNextWgt"
        KeysPrev:        "Panel.KeysPrevWgt"
        KeysNextInGrp:   "Panel.KeysNextRadio"
        KeysPrevInGrp:   "Panel.KeysPrevRadio"
        FgColor:         "LBox.DefFgColor"
        BgColor:         "LBox.DefBgColor"
        Font:    "ttt_se.Font2"
        Pen:     "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor:  "Curs.Cross"
        Index:   7
        X:       5
        Y:       83
        W:       26
        H:       31
        WgtFlags:        0x0001
        Script:  <MOVED TO SCRIPT FILE>
)

(LBox.Compile
        Name:    "ttt_se.winMain.pnlBoard.lbCell8"
        Version: 8
        StartCol: 0x0001
        StartRow: 0x0001
        ColWidth: 0x0032
        RowHeight: 0x0014
        ColNum:         1
        RowNum:         1
        LbKeys: "LBox.KeysDef"
        CellPen: "Win.DefPen"
        TextEditor:     "NMsgEd.EditTEd"
        SbSepW:         4
        SbSepH: 4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        FgColor:        "LBox.DefFgColor"
        BgColor:        "LBox.DefBgColor"
        Font:   "ttt_se.Font2"
        Pen:    "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.Cross"
        Index:  8
        X:      50
        Y:      83
        W:      26
        H:      31
        WgtFlags:       0x0001
        Script: <MOVED TO SCRIPT FILE>
)

(LBox.Compile
        Name:   "ttt_se.winMain.pnlBoard.lbCell9"
        Version: 8
        StartCol: 0x0001
        StartRow: 0x0001
        ColWidth: 0x0032
        RowHeight: 0x0014
        ColNum:         1
        RowNum:         1
        LbKeys: "LBox.KeysDef"
        CellPen: "Win.DefPen"
        TextEditor:     "NMsgEd.EditTEd"
        SbSepW:         4
        SbSepH: 4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        FgColor:        "LBox.DefFgColor"
        BgColor:        "LBox.DefBgColor"
        Font:   "ttt_se.Font2"
        Pen:    "Win.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.Cross"
        Index:  9
        X:      95
        Y:      83
        W:      26
        H:      31
        WgtFlags:       0x0001
        Script: <MOVED TO SCRIPT FILE>
)


(MTEd.Compile
        Name:   "ttt_se.winMain.txtMsgBox"
        Version:8
        Justif: 0x0011
        OptFlags:       0x0002
        LeftMargin:     4
        RightMargin:    4
        LabelFont:      "TEd.DefFont"
        LabelPen:       "Pen.Solid"
        LabelPattern:   "Patt.Empty"
        LabelFgColor:   "TEd.DefFgColor"
        LabelBgColor:   "Color.Transparent"
        InitialText:    "Click on <New Game> to begin."
        SbSepW:         4
        SbSepH:4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        LabelJustif:    0x0041
        FgColor:        "TEd.DefFgColor"
        BgColor:        "TEd.DefBgColor"
        Font:   "TEd.DefFont"
        Pen:    "TEd.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.DefArrow"
        Index:  4
        X:      11
        Y:      212
        W:      311
        H:      81
        RzFlags:        0x0100
        WgtFlags:       0x0001
)

(STEd.Compile
        Name:   "ttt_se.winMain.txtGameState"
        Version:8
        Justif: 0x0011
        OptFlags:       0x0003
        LeftMargin:     4
        RightMargin:    4
        LabelFont:      "TEd.DefFont"
        LabelPen:       "Pen.Solid"
        LabelPattern:   "Patt.Empty"
        LabelFgColor:   "TEd.DefFgColor"
        LabelBgColor:   "Color.Transparent"
        HSepH:  20
        SbSepW:         4
        SbSepH: 4
        KeysNext:       "Panel.KeysNextWgt"
        KeysPrev:       "Panel.KeysPrevWgt"
        KeysNextInGrp:  "Panel.KeysNextRadio"
        KeysPrevInGrp:  "Panel.KeysPrevRadio"
        Label:  "Game State:"
        LabelJustif:    0x0041
        FgColor:        "TEd.DefFgColor"
        BgColor:        "TEd.DefBgColor"
        Font:   "TEd.DefFont"
        Pen:    "TEd.DefPen"
        Pattern: "Patt.Empty"
        Cursor: "Curs.DefArrow"
        Index:  5
        X:      331
        Y:      137
        W:      96
        H:      26
        RzFlags:        0x0100
        WgtFlags:       0x0001
        Script: <MOVED TO SCRIPT FILE>
)
```

Open Interface Script File for Tic Tac Toe

On "winMain" object:

```
on event WIN_OPENED

    //Collect some useful pointers
    winMainptr = WGT_GetWin(SELF);

    txtGameStateptr = WIN_GetNamedWgt(winMainptr,"txtGameState");

    txtMsgBoxptr = WIN_GetNamedWgt(winMainptr,"txtMsgBox");

    //Initialize text areas
    TED_SetStr(txtGameStateptr, "SYSTEM");

    TED_SetStr(txtMsgBoxptr, "Processing...");

    //Initialize Nexpert
    NOIR_RestartSession();

    //Initialize Cells
    lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell1");
    LBOX_SetCellString(lbCellptr,1,1," ");

    lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell2");
    LBOX_SetCellString(lbCellptr,1,1," ");

    lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell3");
    LBOX_SetCellString(lbCellptr,1,1," ");

    lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell4");
    LBOX_SetCellString(lbCellptr,1,1," ");

    lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell5");
    LBOX_SetCellString(lbCellptr,1,1," ");

    lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell6");
    LBOX_SetCellString(lbCellptr,1,1," ");

    lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell7");
    LBOX_SetCellString(lbCellptr,1,1," ");

    lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell8");
    LBOX_SetCellString(lbCellptr,1,1," ");

    lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell9");
    LBOX_SetCellString(lbCellptr,1,1," ");

// Enable <You Go First> Button
    WGT_Enable(WIN_GetNamedWgt(winMainptr, "btnYouGoFirst"));
```

```
   //Set text areas for play
   TED_SetStr(txtMsgBoxptr, "Click on any cell to make your first move, or... Click on the
          <You Go First> button if you want me to go first.");

   TED_SetStr(txtGameStateptr, "PLAY");

end event //WIN_OPENED
```

On “lbCell” object:

```
on event LBOX_CELLCLICKED
   //Collect some useful pointers
   winMainptr = WGT_GetWin(SELF);

   txtGameStateptr = WIN_GetNamedWgt(winMainptr,"txtGameState");

   txtMsgBoxptr = WIN_GetNamedWgt(winMainptr,"txtMsgBox");


   //Just in case the User is faster than the system
   while(TED_GetStr(txtGameStateptr) == "SYSTEM")
   {
     LBOX_UnselectCell(SELF,1,1);
     return;
   }

   // Gray out <You Go First> Button
   WGT_Disable(WIN_GetNamedWgt(winMainptr, "btnYouGoFirst"));

   //Make sure the game is still in progress
   while(TED_GetStr(txtGameStateptr) != "PLAY")
   {
     TED_SetStr(txtMsgBoxptr,"Get over it... because the game is. Click on <NewGame>
         to play again.");
     LBOX_UnselectCell(SELF,I,1);
     return;
   }

   //Validate User Move
   currVal = LBOX_GetCellString(SELF,1,1);
   if(currVal != " "){
     TED_SetStr(txtMsgBoxptr, "Sorry, that cell is already in use...please try again.");
   }
   else{
   //Take Control and process User's move
     TED_SetStr(txtGameStateptr, "SYSTEM");

     TED_SetStr(txtMsgBoxptr, "Processing...");

  //Update board with User's move
     LBOX_SetCellString(SELF,1,1,"X");
   //Update and run the NEXPERT game engine
      NOIR_Volunteer(NOIR_GetAtomId("Cell1.Val", NXP_ATYPE_SLOT), NXP_DESC_INT, 1,
          NXP_VSTRAT_VOLFWRD);

      NOIR_SendMessage("mthdUpdate", NOIR_GetAtomId("Sums", NXP_ATYPE_CLASS), "");

      NOIR_Suggest(NOIR_GetAtomId("hypMakeAMove", NXP_ATYPE_HYPO),
 NXP_SPRIO_SUG);

      NOIR_Knowcess();

      NOIR_ProcessForm(winMainptr);

   //Handle Game State
      NOIR_UpdateWgt(txtGameStateptr);

      if (TED_GetStr(txtGameStateptr) == "PLAY")

         TED_SetStr(txtMsgBoxptr, "Your move.");

      else {
         if (TED_GetStr(txtGameStateptr) == "GAME WINS")

            TED_SetStr(txtMsgBoxptr, "Tic Tac Toe, Three in a Row... You lose, I win! You will have to
                     try again (click on <New Game>).");

         else {
            if (TED_GetStr(txtGameStateptr) == "DRAW")

            TED_SetStr(txtMsgBoxptr, "I guess the cat got this one... Click on <New Game>
                     to play again.");

            else {
               if (TED_GetStr(txtGameStateptr) == "USER WINS")

                  TED_SetStr(txtMsgBoxptr, "I don't believe it, YOU WON!!!...Quick, click on
                     <New Game>.");
            }

         }

      }

   } //end else process the User's move

  LBOX_UnselectCell(SELF,1,1);

end event //LBOX_CELLCLICKED


on event NOIR_PROCESSFORM

  //Test to see if the NEXPERT game engine placed an O in this cell
  currVal = LBOX_GetCellString(SELF,1,1);
  if(currVal == " "){

     nxpCellVal = NOIR_GetIntVal(NOIR_GetAtomId("Cell1.Val", NXP_ATYPE_SLOT));

     if(nxpCellVal == -1)

        LBOX_SetCellString(SELF,1,1,"O");

  }

end event  // NOIR_PROCESSFORM
```

On “btnNewGame” Object:

```
on event TBUT_HIT
  //Collect some useful pointers
  winMainptr = WGT_GetWin(SELF);

  txtGameStateptr = WIN_GetNamedWgt(winMainptr,"txtGameState");

  txtMsgBoxptr = WIN_GetNamedWgt(winMainptr,"txtMsgBox");

  //Initialize text areas
  TED_SetStr(txtGameStateptr, "SYSTEM");

  TED_SetStr(txtMsgBoxptr, "Processing...");

  //Initialize Nexpert
  NOIR_RestartSession();

  //Initialize Cells
  lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell1");
  LBOX_SetCellString(lbCellptr,1,1," ");

  lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell2");
  LBOX_SetCellString(lbCellptr,1,1," ");

  lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell3");
  LBOX_SetCellString(lbCellptr,1,1," ");

  lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell4");
  LBOX_SetCellString(lbCellptr,1,1," ");

  lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell5");
  LBOX_SetCellString(lbCellptr,1,1," ");

  lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell6");
  LBOX_SetCellString(lbCellptr,1,1," ");

  lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell7");
  LBOX_SetCellString(lbCellptr,1,1," ");
  lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell8");
  LBOX_SetCellString(lbCellptr,1,1," ");

  lbCellptr = WIN_GetNamedWgt(winMainptr,"lbCell9");
  LBOX_SetCellString(lbCellptr,1,1," ");

  // Enable <You Go First> Button
  WGT_Enable(WIN_GetNamedWgt(winMainptr, "btnYouGoFirst"));

  //Set text areas for play
  TED_SetStr(txtMsgBoxptr, "Click on any cell to make your first move, or... Click on the
       <You Go First> button if you want me to go first.");

  TED_SetStr(txtGameStateptr, "PLAY");

end event  //TBUT_HIT
```

On “btnQuit” Object:

```
on event TBUT_HIT

  WIN_Terminate(WGT_GetWin(SELF));

end event  //TBUT_HIT
```

On “btnYouGoFirst” Object:

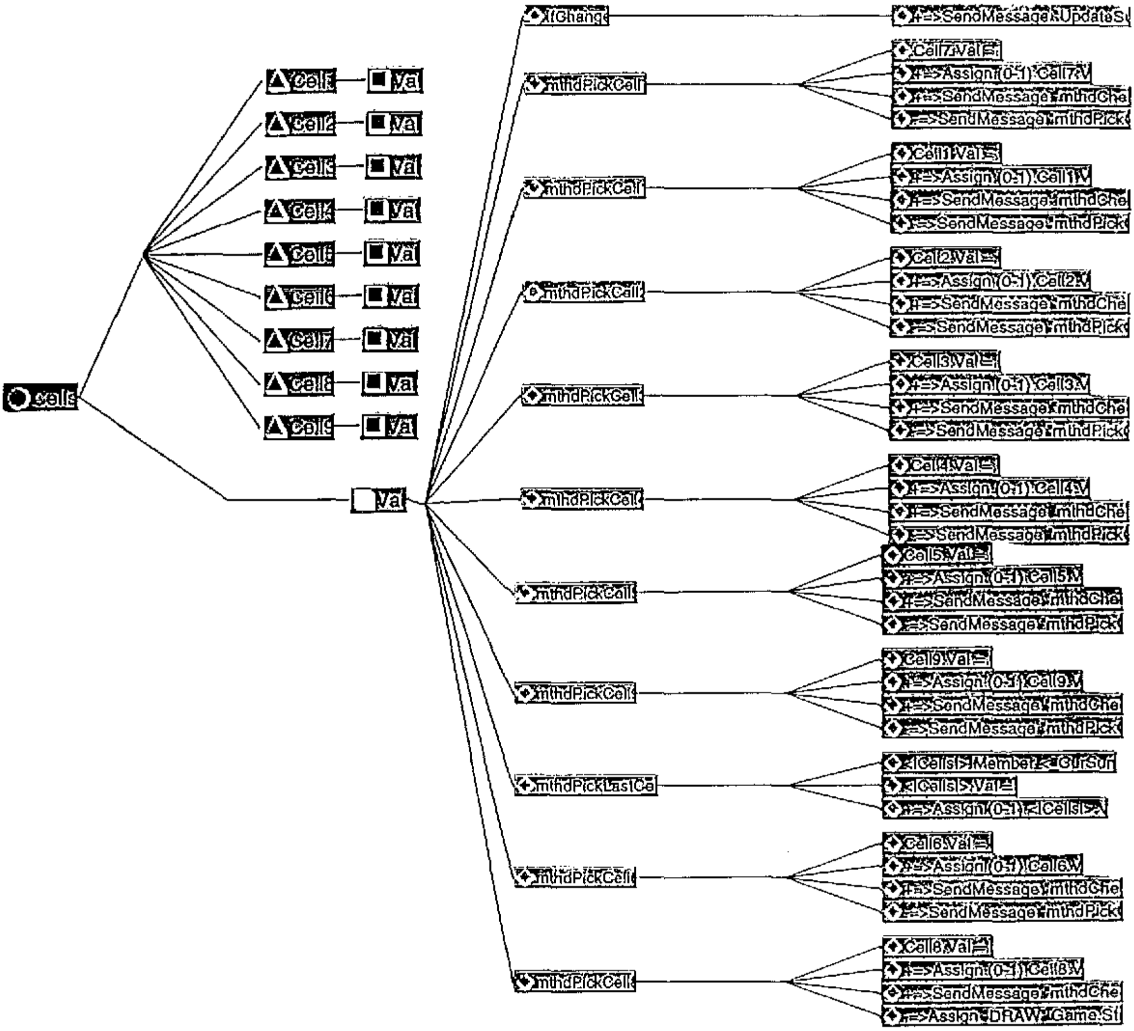
```
on event TBUT_HIT

  //Collect some useful pointers
  winMainptr = WGT_GetWin(SELF);

  txtGameStateptr = WIN_GetNamedWgt(winMainptr,"txtGameState");

  txtMsgBoxptr = WIN_GetNamedWgt(winMainptr,"txtMsgBox");

  // Gray out <You Go First> Button
  WGT_Disable(WIN_GetNamedWgt(winMainptr, "btnYouGoFirst"));

  //Take Control and process User's move
  TED_SetStr(txtGameStateptr, "SYSTEM");

  TED_SetStr(txtMsgBoxptr, "Processing...");

  //Update and run the NEXPERT game engine
  NOIR_SendMessage("mthdUpdate", NOIR_GetAtomId("Sums", NXP_ATYPE_CLASS), "");

  NOIR_Suggest(NOIR_GetAtomId("hypMakeAMove", NXP_ATYPE_HYPO), NXP_SPRIO_SUG);

  NOIR_Knowcess();
```

```
   NOIR_ProcessForm(winMainptr);

   //Handle Game State
   NOIR_UpdateWgt(txtGameStateptr);

   TED_SetStr(txtMsgBoxptr, "Your move.");

end event  //TBUT_HIT
```

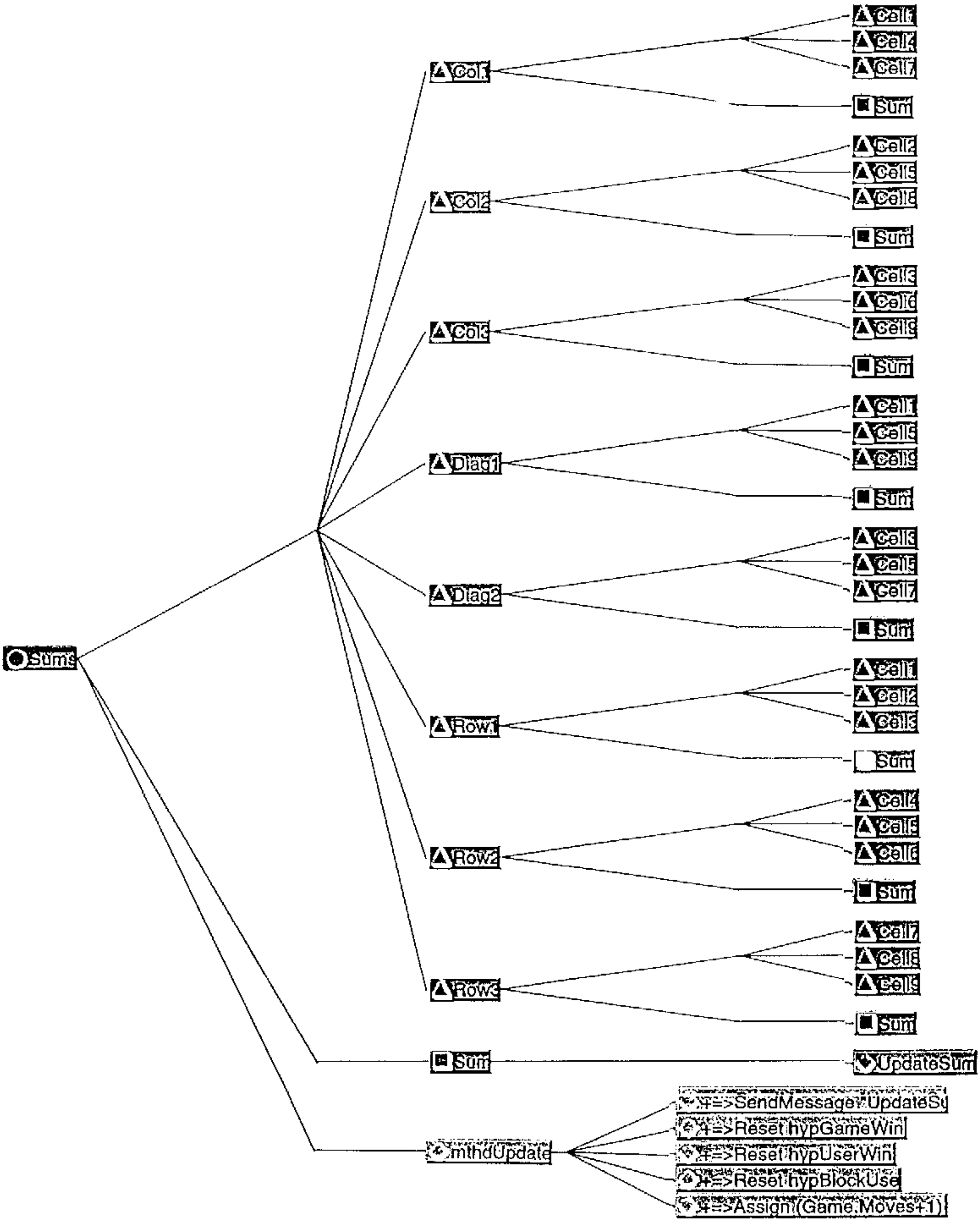
On “txtGameState” Object:

```
on event INITIALIZE

   NOIR_LinkTextEdit(SELF, NOIR_GetAtomId("Game.State", NXP_ATYPE_SLOT), I);

                                        end event  // INITIALIZE
```

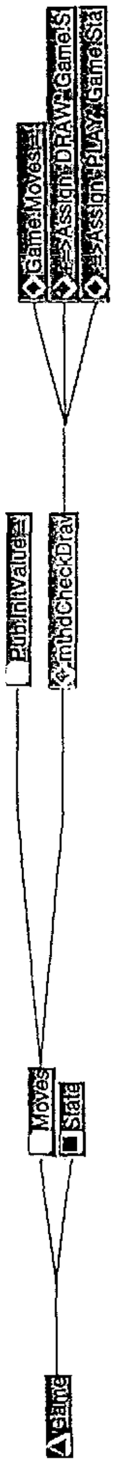
Nexpert Object Graphs and File Excerpts for Tic Tac Toe (TTT.KB)



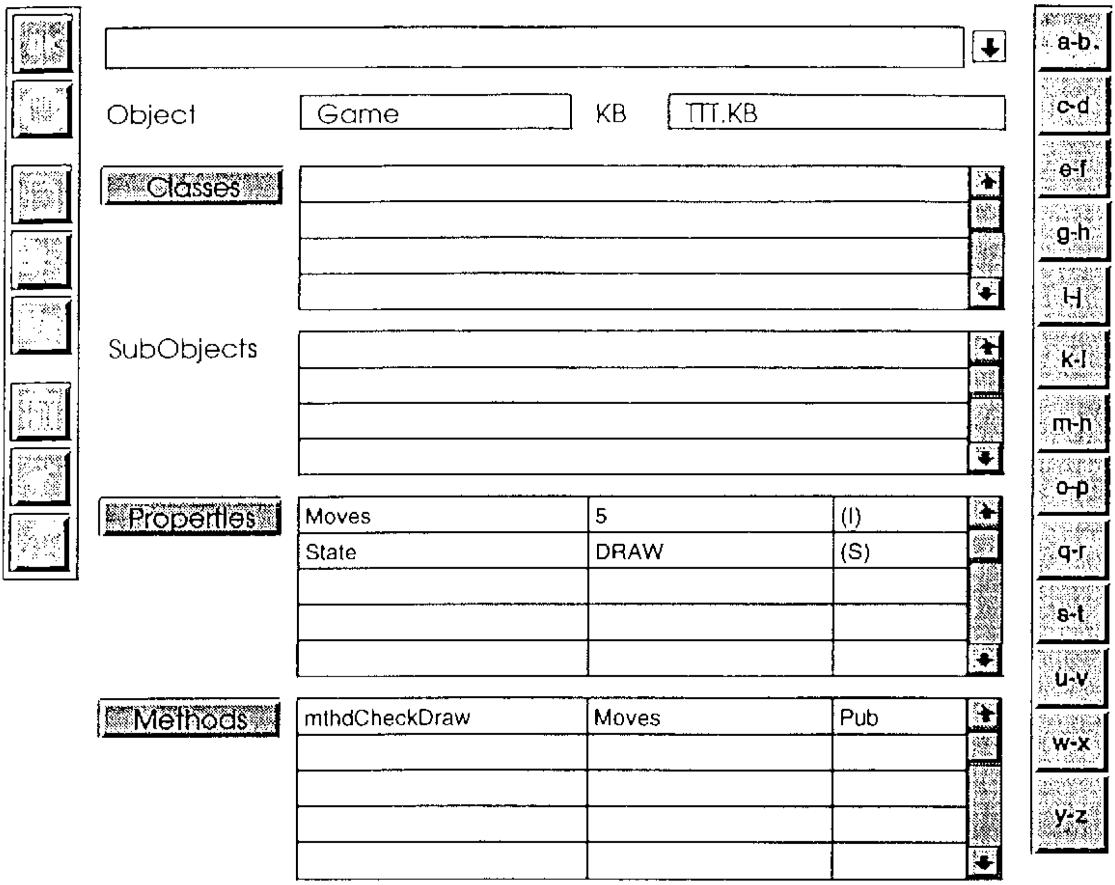
Class-Object Hierarchy for the “Cells” Class



Class-Object Hierarchy for the "Sums" Class



Class-Object Hierarchy for the “Game” Object

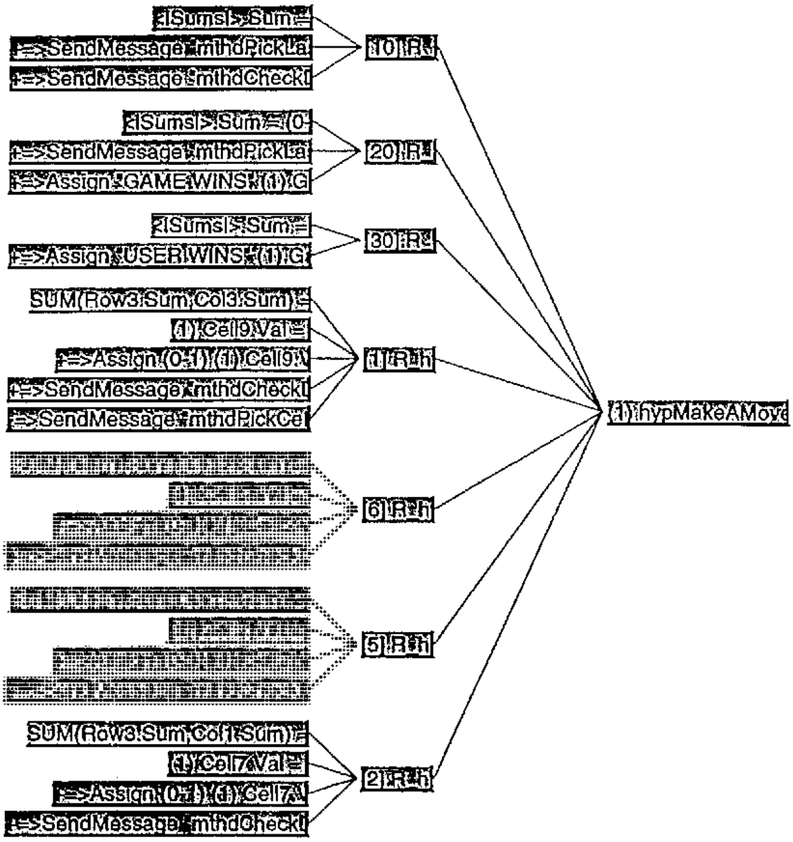


Object Game KB TTT.KB

Moves	5	(I)
State	DRAW	(S)

mthdCheckDraw	Moves	Pub

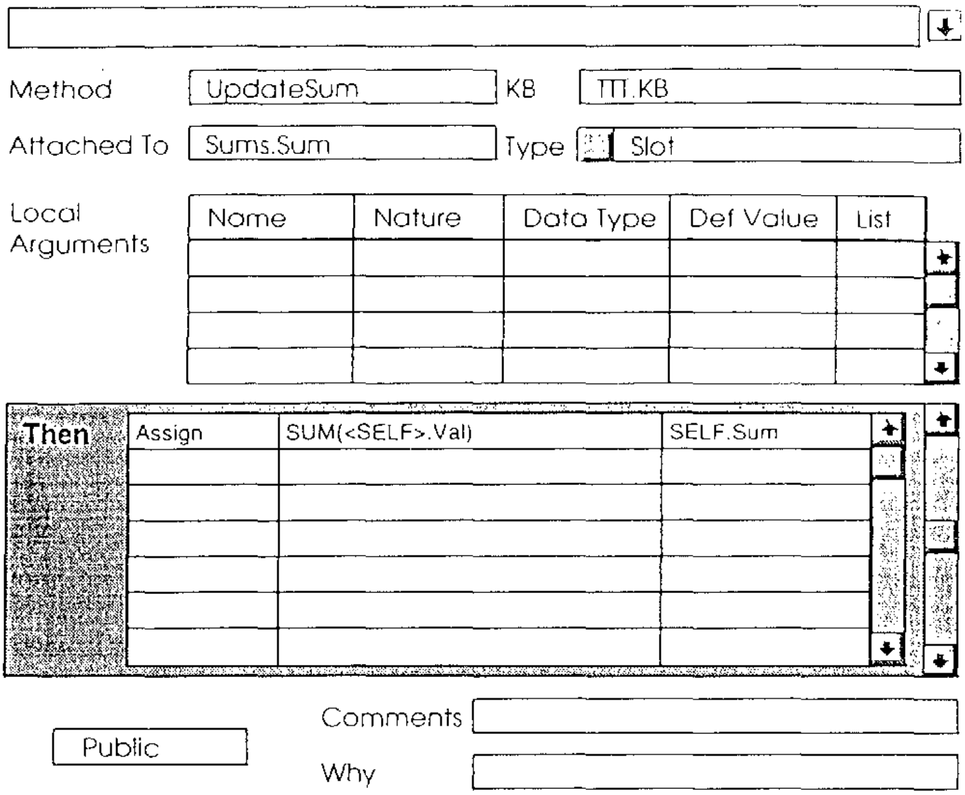
Typical Object Dialog in Nexpert Object



Rule Graph

Rules Listing:

```
(@RULE=         R_hypUserWins
        @INFCAT=30;
        (@LHS=
                (=      (<|Sums|>.Sum)  (3))
        )
        (@HYPO=         hypMakeAMove)
        (@RHS=
                (Assign ("USER WINS") (Game.State))
        )
)
(@RULE=         R_hypGameWins
        @INFCAT=20;
        (@LHS=
                (=      (<|Sums|>.Sum)  ((0-2)))
        )
        (@HYPO=         hypMakeAMove)
        (@RHS=
                (SendMessage    ("mthdPickLastCell")      (@TO=|Cells|.Val;@ARG1=<|Sums|>;))
                (Assign ("GAME WINS")           (Game.State))
        )
)
(@RULE=         R_hypBlockUser
        @INFCAT=10;
        (@LHS=
                (=      (<|Sums|>.Sum)  (2))
        )
        (@HYPO=         hypMakeAMove)
        (@RHS=
                (SendMessage    ("mthdPickLastCell")    (@TO=|Cells|.Val;@ARG1=<|Sums|>;))
                (SendMessage    ("mthdCheckDraw")       (@TO=Game.Moves;))
        )
)
(@RULE=         R_hypBustAWedge1A
        @INFCAT=6;
        (@LHS=
                (=      (SUM(Diag1.Sum,Diag2.Sum))      (0))
                (=      (Cell5.Val)     (0))
        )
        (@HYPO=         hypMakeAMove)
        (@RHS=
                (Assign ((0-1))  (Cell5.Val))
                (SendMessage    ("mthdCheckDraw")       (@TO=Game.Moves;))
        )
)
(@RULE=         R_hypBustAWedge1B
        @INFCAT=5;
        (@LHS=
                (=      (SUM(Diag1.Sum,Diag2.Sum))      (0))
                (=      (Cell2.Val)     (0))
        )
        (@HYPO=         hypMakeAMove)
        (@RHS=
                (Assign ((0-1))  (Cell2.Val))
                (SendMessage    ("mthdCheckDraw")       (@TO=Game.Moves;))
        )
)
(@RULE=         R_hypBustAWedge4
        @INFCAT=2;
        (@LHS=
                (=      (SUM(Row3.Sum,Col1.Sum))        (2))
                (=      (Cell7.Val)     (0))
        )
        (@HYPO=         hypMakeAMove)
        (@RHS=
                (Assign ((0-1))  (Cell7.Val))
                (SendMessage    ("mthdCheckDraw")       (@TO=Game.Moves;))
        )
)
(@RULE=         R_hypBustAWedge5
        (@LHS=
                (=      (SUM(Row3.Sum,Col3.Sum))        (2))
        (=      (Cell9.Val)      (0))
    )
    (@HYPO=     hypMakeAMove)
    (@RHS=
        (Assign ((0-1))  (Cell9.Val))
        (SendMessage   ("mthdCheckDraw")       (@TO=Game.Moves;))
    )
    (@EHS=
        (SendMessage   ("mthdPickCell5")       (@TO=Cell5.Val;))
    )
)
```



Method: UpdateSum

KB: TTT.KB

Attached To: Sums.Sum

Type: Slot

Local Arguments

Name	Nature	Data Type	Def Value	List

Then			
	Assign	SUM(<SELF>.Val)	SELF.Sum

Typical Method Dialog in Nexpert Object

Methods Listing:

```
(@METHOD=   IfChange
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@RHS=
                (SendMessage    ("UpdateSum")   (@TO=<|Sums|>.Sum;))
        )
)
(@METHOD=   mthdCheckDraw
        (@ATOMID=Game.Moves;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=        (Game.Moves)    (5))
        )
        (@RHS=
                (Assign  ("DRAW")        (Game.State))
        )
        (@EHS=
                (Assign ("PLAY")        (Game.State))
        )
)
(@METHOD=   mthdPickCell1
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=        (Cell1.Val)       (0))
        )
        (@RHS=
                (Assign ((0-1))  (Cell1.Val))
                (SendMessage    ("mthdCheckDraw")       (@TO=Game.Moves;))
        )
        (@EHS=
                (SendMessage    ("mthdPickCell3")       (@TO=Cell3.Val;))
        )
)
(@METHOD=   mthdPickCell2
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=        (Cell2.Val)       (0))
        )
        (@RHS=
                (Assign ((0-1))  (Cell2.Val))
                (SendMessage    ("mthdCheckDraw")       (@TO=Game.Moves;))
        )
        (@EHS=
                (SendMessage    ("mthdPickCell4")       (@TO=Cell4.Val;))
        )
)
(@METHOD=   mthdPickCell3
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=        (Cell3.Val)       (0))
        )
        (@RHS=
                (Assign ((0-1))  (Cell3.Val))
                (SendMessage    ("mthdCheckDraw")       (@TO=Game.Moves;))
        )
        (@EHS=
                (SendMessage    ("mthdPickCell7")       (@TO=Cell7.Val;))
        )
)
(@METHOD=   mthdPickCell4
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=        (Cell4.Val)       (0))
        )
        (@RHS=
                (Assign ((0-1))  (Cell4.Val))
                (SendMessage    ("mthdCheckDraw")      (@TO=Game.Moves;))
        )
        (@EHS=
                (SendMessage    ("mthdPickCell6")      (@TO=Cell6.Val;))
        )
)
(@METHOD=   mthdPickCell5
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=        (Cell5.Val)       (0))
        )
        (@RHS=
                (Assign ((0-1))  (Cell5.Val))
                (SendMessage    ("mthdCheckDraw")      (@TO=Game.Moves;))
        )
        (@EHS=
                (SendMessage    ("mthdPickCell1")      (@TO=Cell1.Val;))
        )
)
(@METHOD=   mthdPickCell6
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=        (Cell6.Val)       (0))
        )
        (@RHS=
                (Assign ((0-1))  (Cell6.Val))
                (SendMessage    ("mthdCheckDraw")      (@TO=Game.Moves;))
        )
        (@EHS=
                (SendMessage    ("mthdPickCell8")      (@TO=Cell8.Val;))
        )
)
(@METHOD=   mthdPickCell7
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=        (Cell7.Val)       (0))
        )
        (@RHS=
                (Assign ((0-1))  (Cell7.Val))
                (SendMessage    ("mthdCheckDraw")      (@TO=Game.Moves;))
        )
        (@EHS=
                (SendMessage    ("mthdPickCell9")      (@TO=Cell9.Val;))
        )
)
(@METHOD=   mthdPickCell8
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=          (Cell8.Val)         (0))
        )
        (@RHS=
                (Assign  ((0-1))   (Cell8.Val))
                (SendMessage     ("mthdCheckDraw")         (@TO=Game.Moves;))
        )
        (@EHS=
                (Assign  ("DRAW")          (Game.State))
        )
)
(@METHOD=   mthdPickCell9
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (=          (Cell9.Val)         (0))
        )
        (@RHS=
                (Assign  ((0-1))   (Cell9.Val))
                (SendMessage     ("mthdCheckDraw")         (@TO=Game.Moves;))
        )
        (@EHS=
                (SendMessage     ("mthdPickCell2")         (@TO=Cell2.Val;))
        )
)
(@METHOD=   mthdPickLastCell
        (@ATOMID=Cells.Val;@TYPE=SLOT;)
        (@ARG1=_CurSum;@NATURE=Object;@LIST;)
        (@FLAGS=PUBLIC;)
        (@LHS=
                (Member           (<|Cells|>)          (<_CurSum>))
                (=          (<|Cells|>.Val)    (0))
        )
        (@RHS=
                (Assign  ((0-1))   (<|Cells|>.Val))
        )
)
(@METHOD=   mthdUpdate
        (@ATOMID=Sums;@TYPE=CLASS;)
        (@FLAGS=PUBLIC;)
        (@RHS=
                (SendMessage     ("UpdateSum")    (@TO=<|Sums|>.Sum;))
                (Assign  ((Game.Moves+1))         (Game.Moves))
                (Reset    (hypMakeAMove))
        )
)
(@METHOD=   UpdateSum
        (@ATOMID=Sums.Sum;@TYPE=SLOT;)
        (@FLAGS=PUBLIC;)
        (@RHS=
                (Assign  (SUM(<SELF>.Val))       (SELF.Sum))
        )
)
```

APPENDIX G

SA2VB.EXE Bridge Listings and Sample Results

Source Code for SA2VB.C

```
#include <stdio.h>
#include <ctype.h>
#include "SA-VB.h"
/*#include <strings.h>*/

/* global variables */
FILE *infile; /* to point to the input file */
FILE *outfile; /* to point to the output file */
int COUNTER = 1;

// doControl (recursively) processes CONTROL statements
int doControl(FILE *infile,FILE *outfile){

        char nextword[256];
        char caption[256];
        char name[256];
        char type[256];
        char style[256];
        char count[4];
        int datum, len, flag = 0;

// Scan for CONTROL to process or END to bail
                while (fscanf(infile, "%s", &nextword) != EOF){
                        if(!strcmp(nextword,"CONTROL"))
                                break;
                        else{
                                if(!strcmp(nextword,"END"))
                                        return 0;
                        }
                }

// Pull out the Caption/Text, based on ", as the delimiter
        strcpy(caption,"");
        while(fscanf(infile, "%s", &nextword) != EOF){
                if(strstr(nextword,"\",") == NULL){
                        strcat(caption,nextword);    // keep parsing the caption
                        strcat(caption," ");
                }
                else{
                        len = strlen(nextword);       // lose the comma and break
                        strncat(caption,nextword,len-1);
                        break;
                }
        }

// Grab the object name, checking for illegal names
        strcpy(name,"");
        fscanf(infile, "%s", &nextword);
        len = strlen(nextword);
        strncat(name,nextword,len-1);

// set flag if name begins with a non-alpha, so we can create a good name later
        if(!isalpha(name[0])){
                flag = 1;
        }

// Grab the type of the object
        strcpy(type,"");
        fscanf(infile, "%s", &nextword);
        len = strlen(nextword);
        strncat(type,nextword,len-1);

//Check the flag and generate a unique name, if appropriate
        if(flag){
                itoa(COUNTER,&count,10);
                strcpy(name,"SA2VB_");
                strcat(name,type);
                strcat(name,count);
                flag = 0;
        }

// Grab the style info
        strcpy(style,"");
 fscanf(infile, "%s", &nextword);
        len = strlen(nextword);
        strncat(style,nextword,len-1);

// Set up a COMMAND_BUTTON
        if(!strcmp(type,"BUTTON") && strstr(style,"PUSHBUTTON")){
                fprintf(outfile,"  Begin CommandButton %s\n",name);
                fprintf(outfile,"    Caption\t=\t%s\n",caption);
                //HANDLE STYLE OPTIONS
                if(strstr(style,"DEFPUSHBUTTON")){
                        fprintf(outfile,"    Default\t=\t-1  'True\n");
                }
        }
        else{

// Set up a TEXT_EDIT_BOX
                if(!strcmp(type,"EDIT")){
```

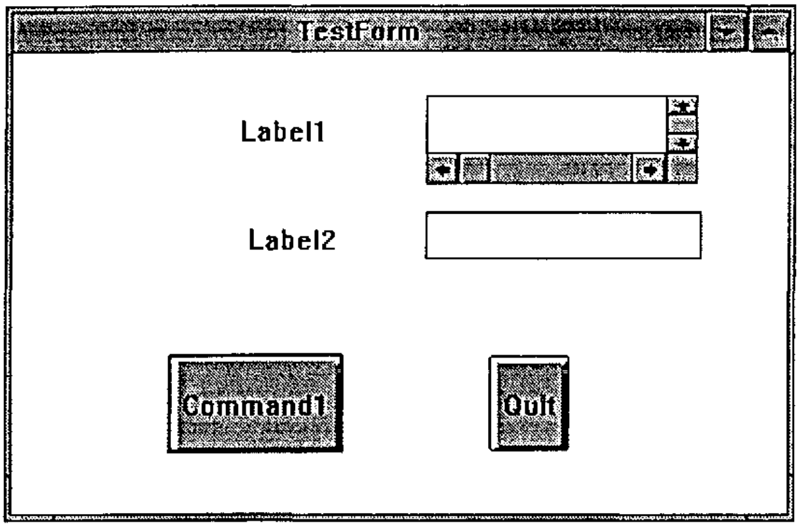
```
            fprintf(outfile," Begin TextBox %s\n",name);
            fprintf(outfile,"  Text\t=\t%s\n",caption);
            //HANDLE STYLE OPTIONS
            if(strstr(style,"VSCROLL") && strstr(style,"HSCROLL")){
                fprintf(outfile,"  ScrollBars\t=\t3  'Both\n");
            }
            else {
                if(strstr(style,"HSCROLL")){
                    fprintf(outfile,"  ScrollBars\t=\t1  'Horizontal\n");
                }
                else {
                    if(strstr(style,"VSCROLL")){
                        fprintf(outfile,"  ScrollBars\t=\t2  'Vertical\n");
                    }
                }
            }
            if(strstr(style,"MULTILINE")){
                fprintf(outfile,"  MultiLine\t=\t-1  'True\n");
            }
            if(strstr(style,"RIGHT")){
                fprintf(outfile,"  Alignment\t=\t1  'Right Justify\n");
            }
            if(strstr(style,"CENTER")){
                fprintf(outfile,"  Alignment\t=\t2  'Center\n");
            }
            if(!strstr(style,"BORDER")){
                fprintf(outfile,"  BorderStyle\t=\t0  'None\n");
            }
        }
        else{
// Set up a LABEL
            if(!strcmp(type,"STATIC")){
                fprintf(outfile," Begin Label %s\n",name);
                fprintf(outfile,"  Caption\t=\t%s\n",caption);
                //HANDLE STYLE OPTIONS
                if(strstr(style,"RIGHT")){
                    fprintf(outfile,"  Alignment\t=\t1  'Right Justify\n");
                }
                if(strstr(style,"CENTER")){
                    fprintf(outfile,"  Alignment\t=\t2  'Center\n");
                }
            }
            else{
// Relinquish processing to the next object
                COUNTER++;
                doControl(infile,outfile);
                return 0;
            }
        }
    }
```

```
// If we're still here, that means it's time to do the coordinates
        fscanf(infile, "%s", &nextword);
        datum = FACTOR * atoi(nextword);
        fprintf(outfile,"   Left\t=\t%d\n",datum);

        fscanf(infile, "%s", &nextword);
        datum = FACTOR * atoi(nextword);
        fprintf(outfile,"   Top\t\t=\t%d\n",datum);

        fscanf(infile, "%s", &nextword);
        datum = FACTOR * atoi(nextword);
        fprintf(outfile,"   Width\t=\t%d\n",datum);

        fscanf(infile, "%s", &nextword);
        datum = FACTOR * atoi(nextword);
        fprintf(outfile,"   Height\t=\t%d\n",datum);

// Close out this object
        fprintf(outfile," End\n");

        COUNTER++;

// Process the next object
        doControl(infile,outfile);

        return 0;

} //end doControl()

void main(int argc, char *argv[]) {

        char nextword[256];
        char caption[256];
        int datum;

        if(argc != 2){  /* verify something was entered as a command line argument */
                printf("You must enter an input filename as a command line argument...exiting.\n");
                exit(0);
        }
infile = fopen(argv[1],"r");

        if(infile == NULL){         /* error-check to confirm successful file opening */
                printf("Can't open input file...exiting.\n");
                exit(0);
        }
        outfile = fopen("out.frm","w");

        if(outfile == NULL){        /* error-check to confirm successful file opening */
                printf("Can't open output file...exiting.\n");
                fclose(infile);
                exit(0);
        }
// Opening Header and Object Name
```

```
        fscanf(infile, "%s", &nextword);
        fprintf(outfile,"VERSION 2.00\nBegin Form %s\n",nextword);

// Verify DIALOG, then scan, calc and output window dimensions
        fscanf(infile, "%s", &nextword);
        if(strcmp(nextword,"DIALOG")){
                printf("Couldn't find DIALOG in input file...exiting.\n");
                fclose(infile);
                fclose(outfile);
                exit(0);
        }
        fscanf(infile, "%s", &nextword);
        datum = FACTOR * atoi(nextword);
        fprintf(outfile,"  Left\t\t=\t%d\n",datum);

        fscanf(infile, "%s", &nextword);
        datum = FACTOR * atoi(nextword);
        fprintf(outfile,"  Top\t\t=\t%d\n",datum);

        fscanf(infile, "%s", &nextword);
        datum = (FACTOR * atoi(nextword)) + 60;
        fprintf(outfile,"  Width\t\t=\t%d\n",datum);

        fscanf(infile, "%s", &nextword);
        datum = (FACTOR * atoi(nextword)) + 360;
        fprintf(outfile,"  Height\t=\t%d\n",datum);

// Verify STYLE information is present
        fscanf(infile, "%s", &nextword);
        if(!strcmp(nextword,"STYLE")){

// Handle STYLE information
                fscanf(infile, "%s", &nextword);
                if(!strstr(nextword,"BORDER") && !strstr(nextword,"THICKFRAME")){
                        fprintf(outfile,"  BorderStyle\t=\t0  'None\n");
                }
                if(strstr(nextword,"BORDER") && !strstr(nextword,"THICKFRAME")){
                        fprintf(outfile,"  BorderStyle\t=\t1  'Fixed Single\n");
                }
                if(!strstr(nextword,"SYSMENU")){
                        fprintf(outfile,"  ControlBox\t=\t0  'False\n");
                }
                if(!strstr(nextword,"MAXIMIZEBOX")){
                        fprintf(outfile,"  MaxButton\t=\t0  'False\n");
                }
                if(!strstr(nextword,"MINIMIZEBOX")){
                        fprintf(outfile,"  MinButton\t=\t0  'False\n");
                }
        }
// Verify CAPTION is present
        fscanf(infile, "%s", &nextword);
        if(!strcmp(nextword,"CAPTION")){
```

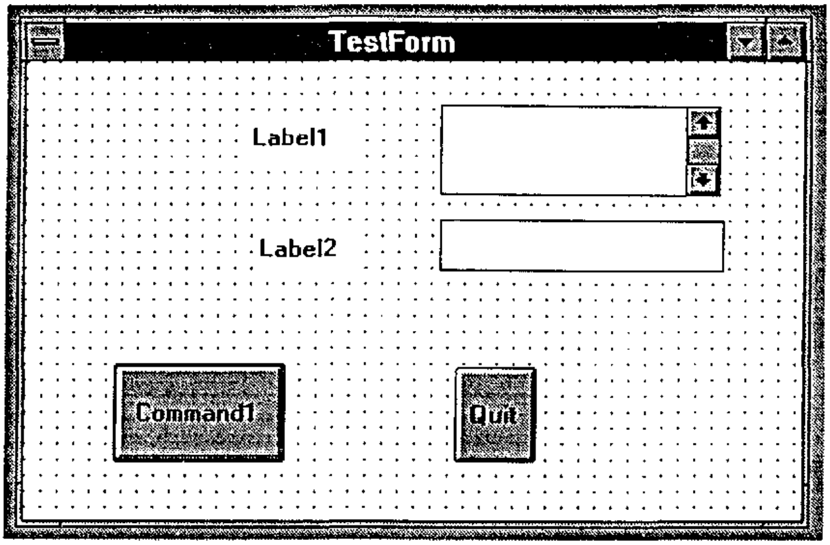
```
// Handle CAPTION
                strcpy(caption,"");
                fscanf(infile, "%s", &nextword);
                strcat(caption,nextword);
                while (fscanf(infile, "%s", &nextword) != EOF){
                        if(strcmp(nextword,"BEGIN")){
                                strcat(caption," ");
                                strcat(caption,nextword);
                        }
                        else
                                break;
                }
                fprintf(outfile,"  Caption\t=\t%s\n",caption);
        }
        doControl(infile,outfile);

        fprintf(outfile,"End\n");
        fclose(infile);
 fclose(outfile);

} /* end main */
```

"TestForm" Screen Designs



"TEST.DLG" from System Architect

```
TestForm DIALOG 48, 51, 192, 117
STYLE WS_TABSTOP|WS_GROUP|WS_BORDER|WS_BORDER|WS_DLGFRAME|
        WS_MINIMIZEBOX|WS_MAXIMIZEBOX|WS_THICKFRAME
CAPTION "TestForm"
BEGIN
        CONTROL "Label1", IDG_LABEL1, STATIC, SS_LEFT, 56, 16, 25, 9
        CONTROL "Label2", IDG_LABEL2, STATIC, SS_LEFT, 58, 44, 25, 9
        CONTROL "", IDG_TEXT1, EDIT,
                WS_TABSTOP|WS_BORDER|WS_BORDER|WS_VSCROLL|WS_HSCROLL|
                ES_MULTILINE|ES_CENTER, 102, 11, 68, 23
        CONTROL "", IDG_TEXT2, EDIT, WS_TABSTOP|WS_BORDER|WS_BORDER|ES_LEFT,
                102, 40, 69, 13
        CONTROL "Command1", IDG_COMMAND1, BUTTON,
                WS_TABSTOP|BS_DEFPUSHBUTTON, 22, 76, 43, 25
        CONTROL "Quit", IDG_COMMAND2, BUTTON, WS_TABSTOP|BS_PUSHBUTTON, 106,
                77, 20, 24
END
```

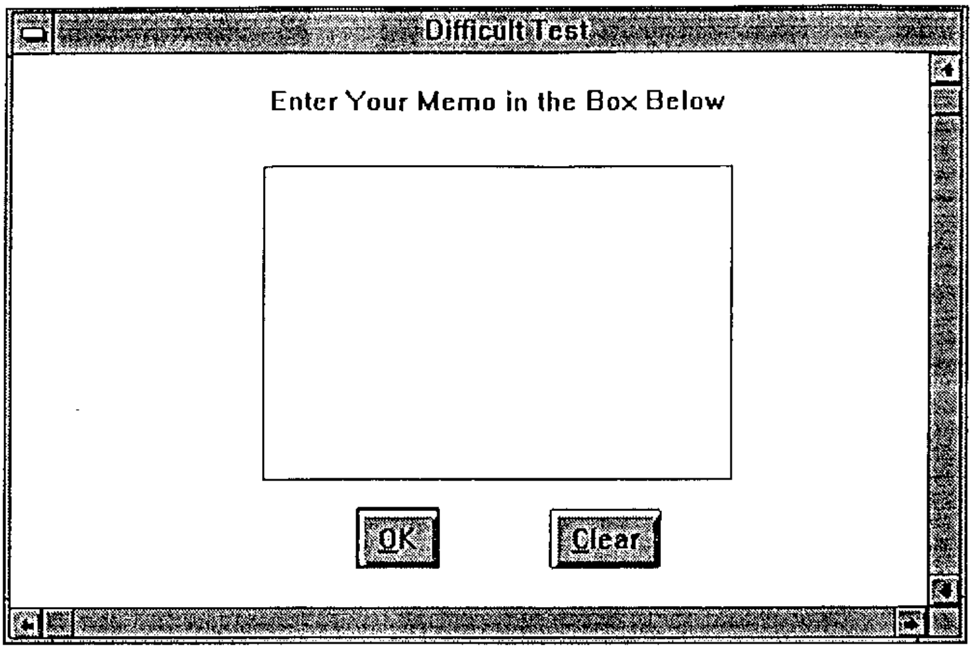


"TEST.FRM" from Visual Basic

```
VERSION 2.00
Begin Form TestForm
 Left           =       1440
 Top            =       1530
 Width          =       5820
 Height         =       3870
 ControlBox     =       0  'False
 Caption        =       "TestForm"
 Begin Label IDG_LABEL1
  Caption       =       "Label1"
  Left          =       1680
  Top           =       480
  Width         =       750
  Height        =       270
 End
 Begin Label IDG_LABEL2
  Caption       =       "Label2"
  Left          =       1740
  Top           =       1320
  Width         =       750
  Height        =       270
 End
 Begin TextBox IDG_TEXT1
  Text          =       ""
  ScrollBars    =       3  'Both
  MultiLine     =       -1  'True
  Alignment     =       2  'Center
```

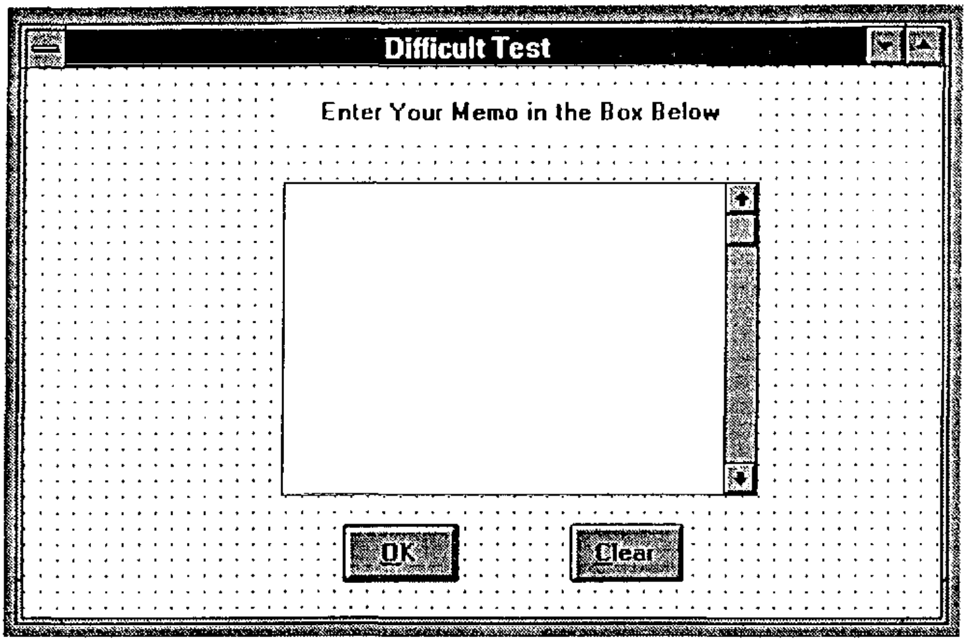
```
     Left        =       3060
     Top         =       330
     Width       =       2040
     Height      =       690
   End
   Begin TextBox IDG_TEXT2
     Text        =       ""
     Left        =       3060
     Top         =       1200
     Width       =       2070
     Height      =       390
   End
   Begin CommandButton IDG_COMMAND1
     Caption     =       "Command1"
     Default     =       -1  'True
     Left        =       660
     Top         =       2280
     Width       =       1290
     Height      =       750
   End
   Begin CommandButton IDG_COMMAND2
     Caption     =       "Quit"
     Left        =       3180
     Top         =       2310
     Width       =       600
     Height      =       720
   End
End
```

“Difficult Test” Screen Designs



“TEST4.DLG” from System Architect

```
hardtest DIALOG 14, 17, 227, 140
STYLE WS_BORDER|WS_BORDER|WS_DLGFRAME|WS_VSCROLL|WS_HSCROLL|
        WS_SYSMENU|WS_THICKFRAME
CAPTION "Difficult Test"
BEGIN
        CONTROL "&OK", IDG_BUTTON1, BUTTON, WS_TABSTOP|BS_DEFPUSHBUTTON,
                78, 115, 29, 15
        CONTROL "&Clear", IDG_BUTTON2, BUTTON, WS_TABSTOP|BS_PUSHBUTTON, 134,
                115, 28, 15
        CONTROL "Enter Your Memo in the Box Below", IDG_TITLE, STATIC,
                SS_NOPREFIX|SS_CENTER, 63, 8, 117, 9
        CONTROL "", IDG_MEMO, EDIT,
                WS_TABSTOP|WS_BORDER|WS_BORDER|ES_AUTOVSCROLL|ES_MULTILINE
                |ES_LEFT, 63, 29, 117, 79
END
```



"TEST4.FRM" from Visual Basic

```
VERSION 2.00
Begin Form hardtest
 Left          =       420
 Top           =       510
 Width         =       6870
 Height        =       4560
 MaxButton     =       0  'False
 MinButton     =       0  'False
 Caption       =       "Difficult Test"
 Begin CommandButton IDG_BUTTON1
  Caption      =       "&OK"
  Default      =       -1  'True
  Left         =       2340
  Top          =       3450
  Width        =       870
  Height       =       450
 End
 Begin CommandButton IDG_BUTTON2
  Caption      =       "&Clear"
  Left         =       4020
  Top          =       3450
  Width        =       840
  Height       =       450
 End
 Begin Label IDG_TITLE
  Caption      =       "Enter Your Memo in the Box Below"
  Alignment    =       2  'Center
```

```
   Left         =       1890
   Top          =       240
   Width        =       3510
   Height       =       270
  End
  Begin TextBox IDG_MEMO
   Text         =       ""
   ScrollBars   =       2  'Vertical
   MultiLine    =       -1  'True
   Left         =       1890
   Top          =       870
   Width        =       3510
   Height       =       2370
  End
End
```

APPENDIX H

Test Bed (Self) Observation Data Sheets

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict ☐ Synergy	Project: ☐ Customer Svc DB ☐ Tic Tac Toe ☒ Both	Category: User I/F Layout	Date Initially Observed: 3/29/94 23:01 Number of Times Observed: 2
SE Application: ☐ Manual ☒ CASE ☐ N/A	SE Method: ☐ Gane & Sarson ☐ Coad/Yourdon ☒ Both	Tool/Language: ☒ Visual Basic ☒ ObjectVision ☒ Visual C++ ☒ Smart Elements	

Description:
When one uses the screen layout facility of the CASE tool to design the User I/F, the work must be duplicated in the Visual Programming tool/language.

Circumstances:
This problem only surfaces when using the CASE tool's screen layout facility. It would also apply to the use of drawing and charting tools.

Guidance Ideas:
Look for CASE tools that can automatically generate the User I/F "code" in the native tongue of the Visual tool/language, thus remedying this conflict. (Note: I don't know of any yet on the market.)

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict ☐ Synergy	Project: ☐ Customer Svc DB ☒ Tic Tac Toe ☐ Both	Category: Event-Based Design	Date Initially Observed: 4/24/94 16:33 Number of Times Observed: 1
SE Application: ☐ Manual ☐ CASE ☒ N/A	SE Method: ☐ Gane & Sarson ☒ Coad/Yourdon ☐ Both	Tool/Language: ☒ Visual Basic ☐ ObjectVision ☐ Visual C++ ☐ Smart Elements	

Description:
VB only provides "when_changed" event processing on the primary attribute of an object (as determined at the "factory"); this means that changes in other properties, such as the "Tag" property, go unnoticed.

Circumstances:
In the TTT game, the design called for tracking of both the text value ("X", "O", or " ") and its numerical equivalent (1, -1, or 0) for each cell. The original design had the User causing the text value (Primary) to change to "X" and using a when_changed method to update the numerical value (Tag) followed by a when_changed method on the numerical value to update the strategy values (sum of rows, columns and diagonals); this approach had to be altered to let the first change drive both updates.

Guidance Ideas:
When VB is known to be the implementation language, the design should be geared to have only one property per object causing event-based behaviors to execute.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict ☐ Synergy	**Project:** ☐ Customer Svc DB ☒ Tic Tac Toe ☐ Both	**Category:** Event-Based Design	**Date Initially Observed:** 4/24/94 16:31 **Number of Times Observed:** 1
SE Application: ☐ Manual ☐ CASE ☒ N/A	**SE Method:** ☐ Gane & Sarson ☒ Coad/Yourdon ☐ Both	**Tool/Language:** ☒ Visual Basic ☐ ObjectVision ☐ Visual C++ ☐ Smart Elements	

Description:
When using an object-oriented approach, it is common to abstract all system parameters as objects; however, VB only uses objects to represent screen entities (so-called controls). Thus, entities that would normally be represented as objects, other than those visible to the User, must be represented as variables of some type.

Circumstances:
In the VB TTT game, there is a strategy area kept "behind the scenes" for use by the program to determine its next move (the sum of the cell values by Rows, Columns and Diagonals). The original design called for creation of a strategy object to maintain such information. The design had to be altered to use global variables instead.

Guidance Ideas:
When VB is known to be the implementation language, one should avoid the use of objects other than those destined for the User Interface. Another approach (not tested) is to create a Virtual Form to hold objects which will be used internally but never actually displayed to the User; this would in essence "trick" VB into having a collection of objects for use "behind the scenes."

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☐ Conflict ☒ Synergy	**Project:** ☐ Customer Svc DB ☒ Tic Tac Toe ☐ Both	**Category:** Event-based Design	**Date Initially Observed:** 4/24/94 17:21 **Number of Times Observed:** 1
SE Application: ☐ Manual ☐ CASE ☒ N/A	**SE Method:** ☐ Gane & Sarson ☒ Coad/Yourdon ☐ Both	**Tool/Language:** ☒ Visual Basic ☐ ObjectVision ☐ Visual C++ ☐ Smart Elements	

Description:
Once the object-oriented nuances of VB were understood and factored into the design, the implementation of the TTT game in VB went very smoothly. I was able to construct the program incrementally without any noteworthy difficulties.

Circumstances:
The C/Y methodology presumes a certain level of support of both object-oriented constructs and event-driven behaviors, while VB only partially supports them (see related "conflicts").

Guidance Ideas:
Once VB has been chosen as the implementation language, avoid fighting the language; it is better to adapt (limit) the design methodology to those object-oriented/event-driven features supported by VB and use procedural approaches for the balance.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type:
☒ Conflict
□ Synergy

Project:
□ Customer Svc DB
☒ Tic Tac Toe
□ Both

Category:
Event-Based Design

Date Initially Observed:
7/27/94 20:59

Number of Times Observed:
1

SE Application:
□ Manual
□ CASE
☒ N/A

SE Method:
□ Gane & Sarson
☒ Coad/Yourdon
□ Both

Tool/Language:
□ Visual Basic □ ObjectVision
□ Visual C++ ☒ Smart Elements

Description:
Communication of events (via SendMessage) from the User Interface (Open Interface) to the Engine (Nexpert Object) is limited to a single target object. Updating of the sums of the various Rows, Columns and Diagonals required the message to be broadcast to all childran of Sums. To get around this limitation, two Methods were implemented; one attached to the parent Sums class which allowed the Interface to interact at a "single point of contact"; and a more complex one which was in turn activated by the simple one once control was passed from the Interface to the Engine.

Circumstances:
This was a minor incouvience; in fact, once the second, more complex Method was in place, I found more ways to take advantage of it.

Guidance Ideas:
It may actually be preferable to have communication links between major system modules be simple (like a "pinch point"). A single, simple message from one module to the other is easy to follow and debug if problems do arise. The target method can than spawn however complex a set of processes as are required.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type:
☒ Conflict
□ Synergy

Project:
□ Customer Svc DB
☒ Tic Tac Toe
□ Both

Category:
User I/F Layout

Date Initially Observed:
7/27/94 21:16

Number of Times Observed:
1

SE Application:
□ Manual
□ CASE
☒ N/A

SE Method:
□ Gane & Sarson
☒ Coad/Yourdon
□ Both

Tool/Language:
□ Visual Basic □ ObjectVision
□ Visual C++ ☒ Smart Elements

Description:
The design called for a simple Text Edit Box as the mechanism for building the nine cells of the Tic Tac Toe Board. However, the Smart Elements Text Edit object provided a very limited of automatic Events; in particular, it did NOT provide a MouseClick event as called for in the design. Instead, I had to use the more complex List Box object (which is similar to a spreadsheet in nature), limiting it to a single row and a single column. This provided access to the MouseClick event.

Circumstances:
Smart Elements does support the creation of custom widgets, so I could have gone outside of the "built in" widgets and added the MouseClick event to the basic Text Edit object. However, a goal of mine was to use the "off the shelf" widgets and scripting language to implement the design.

Guidance Ideas:
Look beyond the obvious in stretching the featrues of the tool. A 1x1 spreadsheet looks exactly like a Text Edit object and behaves just the design needed it to.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict □ Synergy	Project: □ Customer Svc DB ☒ Tic Tac Toe □ Both	Category: Event-Based Design	Date Initially Observed: 7/27/94 21:27 Number of Times Observed: 1
SE Application: □ Manual □ CASE ☒ N/A	SE Method: □ Gane & Sarson ☒ Coad/Yourdon □ Both	Tool/Language: □ Visual Basic □ ObjectVision □ Visual C++ ☒ Smart Elements	

Description:
Smart Elements does not provide a means for the Engine (Nexpert Object) to send messages back to the Interface (Open Interface). To work around this, each time control is retruned to the Interface I had to have each cell in the Interface query its latest value in the corresponding object in the Engine and update its own current value accordingly.

Circumstances:
Smart Elements does provide a mechanism referred to as "object linking" which causes an object in the User Interface to be continuously updated with the value of a corresponding object in the Engine. Unfortunately, this approach was not usable in this instance because the Engine object used numeric values (-1,0,+1), while the User Interface object required symbolic values (O, ' ', X).

Guidance Ideas:
The workaround used in this case causes a great deal of extra work, since every interface object must be "pulsed" after each call to the Engine. In a full strength application, where performance could be in jeopardy, a more focused messaging system would have to be crafted (e.g., a "black board" object onto which the Engine could place a list of udated Engine objects and which an Interface method could use to udate just those Interface objects whose Engine values had changed. Alternatively, the linking mechanism could have been used "as is" and then have an Interface method convert the numeric value into its symbolic equivalent.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict □ Synergy	Project: □ Customer Svc DB ☒ Tic Tac Toe □ Both	Category: User I/F Layout	Date Initially Observed: 7/27/94 21:47 Number of Times Observed: 1
SE Application: □ Manual □ CASE ☒ N/A	SE Method: □ Gane & Sarson ☒ Coad/Yourdon □ Both	Tool/Language: □ Visual Basic □ ObjectVision □ Visual C++ ☒ Smart Elements	

Description:
The design called for an indexed set of nine Cells for the Tic Tac Toe Board. The Smart Elements User Interface widgets are not directly indexed (as in Visual Basic). This capability improves the referencing of the objects as well as the ability to "clone" the cells without modifying the underlying code.

Circumstances:
Working around this shortcoming was fairly manageable for this simple application; however, a more complex application would suffer greatly.

Guidance Ideas:
Smart Elements provides the ability to create so-called custom widgets. Doing such would provide the ability to incorporate whatever "generic" attributes and behaviors the object should have (a BoardCell in this case) and then create instances of it in the User Interface.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☐ Conflict ☒ Synergy	**Project:** ☐ Customer Svc DB ☒ Tic Tac Toe ☐ Both	**Category:** Event-Based Design	**Date Initially Observed:** 7/27/94 21:58 **Number of Times Observed:** 5
SE Application: ☐ Manual ☐ CASE ☒ N/A	**SE Method:** ☐ Gane & Sarson ☒ Coad/Yourdon ☐ Both	**Tool/Language:** ☐ Visual Basic ☐ ObjectVision ☐ Visual C++ ☒ Smart Elements	

Description:
The Smart Elements Engine (Nexpert Object) fully and naturally supports the object hierarchy and inheritance called for in the design. Extremely generic methods were written and applied at the class level, inherited down to the appropriate child objects and then applied to give the desired result.

Circumstances:
My goal was to not have to tweak the code used in methods based on where the code was attached. The evolution of the application was inspiring. A mere skeleton of the game was used to develop the Methods; when the rest of the game objects were added, the fully implemented game board worked on the FIRST TRY!

Guidance Ideas:
If you are going to use an object-oriented tool to implement an application, Go For The Gold in the design process. Craft Methods that are as generic as possible. Apply them as high in the hierarchy as possible. Take full advantage of classification structures and let the benefits of OOP shine through.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☐ Conflict ☒ Synergy	**Project:** ☐ Customer Svc DB ☒ Tic Tac Toe ☐ Both	**Category:** Function Design	**Date Initially Observed:** 11/21/94 17:30 **Number of Times Observed:** 1
SE Application: ☐ Manual ☐ CASE ☒ N/A	**SE Method:** ☐ Gane & Sarson ☒ Coad/Yourdon ☐ Both	**Tool/Language:** ☐ Visual Basic ☐ ObjectVision ☐ Visual C++ ☐ Smart Elements	

Description:
The rules in Smart Elements are easily visualized in the rules browser. This made it very to implement the desired gaming strategy in the inference engine. Although this program is neither an expert system nor performing any reasoning or inferencing, the rule-based paradigm proved to facilitate the implementation process.

Circumstances:
Implementation of the control logic for deciding the appropriate response to a user selection of a space on the playing board.

Guidance Ideas:
Consider the use of a tool that provides offers an explicit rule-based paradigm, even if the application is not an expert system or does not require inferencing. The rules can be used to expedite control strategy/logic or to explicitly represent the business rules to be followed.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict ☐ Synergy	Project: ☒ Customer Svc DB ☐ Tic Tac Toe ☐ Both	Category: DB Schema	Date Initially Observed: 11/22/94 17:53 Number of Times Observed: 6
SE Application: ☐ Manual ☒ CASE ☐ N/A	SE Method: ☒ Gane & Sarson ☐ Coad/Yourdon ☐ Both	Tool/Language: ☒ Visual Basic ☒ ObjectVision ☐ Visual C++ ☐ Smart Elements	

Description:
Unable to directly create the schema using the CASE tool.

Circumstances:
The versions of the development tools being used were built to support less open databases. In particular, automated schema generation was not available for VB (Access 1.1) or ObjectVision (Paradox 2.x or dbase).

Guidance Ideas:
When using a full-featured CASE tool, be sure to select a database engine that is supported by the CASE tool's schema generator. For example, VB now supports Microsoft Access 2.0 and the System Architect CASE tool can generate a "vanilla" SQL that can be used with minor editing to automatically create the database structures.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict ☐ Synergy	Project: ☒ Customer Svc DB ☐ Tic Tac Toe ☐ Both	Category: DB Schema	Date Initially Observed: 11/23/94 0:07 Number of Times Observed: 12
SE Application: ☐ Manual ☒ CASE ☐ N/A	SE Method: ☒ Gane & Sarson ☐ Coad/Yourdon ☐ Both	Tool/Language: ☒ Visual Basic ☒ ObjectVision ☐ Visual C++ ☐ Smart Elements	

Description:
Difficult to keep CASE representation of the application in synch with changes which evolved during the implementation process.

Circumstances:
The versions of the development tools being used were built to support less open databases. In particular, reverse engineering was not available for VB (Access 1.1) or ObjectVision (Paradox 2.x or dBase).

Guidance Ideas:
1. Settle on design conformity/leniancy rules and design update frequencies and manually synchronize the design in the CASE tool from time to time.
2. Select the CASE tool and database such that reverse engineering can be used to convey changes implemented in the development tool back to the CASE tool.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: □ Conflict ☒ Synergy	Project: ☒ Customer Svc DB □ Tic Tac Toe □ Both	Category: DB Schema	Date Initially Observed: 11/23/94 0:20 Number of Times Observed: 3
SE Application: □ Manual □ CASE ☒ N/A	SE Method: ☒ Gane & Sarson □ Coad/Yourdon □ Both	Tool/Language: □ Visual Basic ☒ ObjectVision □ Visual C++ □ Smart Elements	

Description:
Manual creation of the data schema was extremely easy (about 5 minutes given that the User Interface was already mapped out).

Circumstances:
ObjectVision presumes that the development process will call for creation of the User Interface (called Forms) prior to creation of the database tables. Thus, it has a very sophisticated schema creation engine that assumes that data items grouped on a Form will likely be in the same table, and that their data types and sizes will conform with how they are specified on the Form as well. It then creates a "straw man" data structure which can then be edited for those places where the assumption does not hold.

Guidance Ideas:
Know your tool's presumptions and go with the flow. By simply understanding the expected sequence of development, one can streamline the development process. Conversely, bucking the system can easily cripple an otherwise useful tool. This is not to say that one should use risky or unsatifactory development practices. And, of course, never, never, never would I suggest that one change the problem to suit the tool. However, if one approach is about the same as another, then let the expected synergy with the development tool make the decision. This, in turn, means that someone on the development team must know (or be able to find out) how the tool expects the problem to be tackled.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict □ Synergy	Project: ☒ Customer Svc DB □ Tic Tac Toe □ Both	Category: Function Design	Date Initially Observed: 11/23/94 0:37 Number of Times Observed: 2
SE Application: □ Manual □ CASE ☒ N/A	SE Method: ☒ Gane & Sarson □ Coad/Yourdon □ Both	Tool/Language: ☒ Visual Basic ☒ ObjectVision □ Visual C++ □ Smart Elements	

Description:
I did not find the Gane & Sarson Process Diagramming technique to be expressive enough to take directly to implementation. Ideally, the desired design could be modelled to a fairly detailed level without prior knowledge of the implementation tool, with perhaps a final layer which takes into account particulars of the tool.

Circumstances:
Each of the implementation tools turned out to require quite different designs in certain key areas. The process model in the design was bland, leaving most of the details to be worked out during implementation. Especially troublesome were the built-in "power" features, such as automation of referential integrity in OV, but not in VB, or automated joins in OV, but not in VB. As I look at the working application, I still ponder how to represent the (now proven) design in Gane & Sarson. (!!!! I need to look at this some more to make sure the problem isn't my greenness in using G&S Process Diagrams. If so, I will redirect this one towards how to model better. !!!)

Guidance Ideas:
To accomodate nuances and/or unknowns of the implementation tool, the design must be kept generic (tool-free) down to a point. Then, if the tool and its special needs are known, a layer of specialization can be added. (Note that the Gane & Sarson process modelling technique uses a drill-down approach to specificity, thus making it suitable for this approach.)

APPENDIX I

Peer Observation Data Sheets

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict □ Synergy	Project: KMM - Numerous	Category: Query Design	Date of Interview: 11/23/94 15:45 Frequency of Obsvervation: Often
SE Application: □ Manual □ CASE ☒ N/A	SE Method: Any	Visual Tool: Any	

Description:
There is frequently a disjoint between what is specified in the design of a query and the behavior that is actually implemented. Also a problem in non-visual tools, but with visual tools the wrong behaviors can be implemented much faster.

Circumstances:
When a query is specified, the exact database schema is often not fully established. Sometimes the SQL dialect is also unknown. Thus, the query will likely be specified in a general way, typically using the language of the domain (as opposed to psuedo-code or simplified SQL). When it comes time for the application developer to implement the query, the essence of what was intended is lost. The query does not behave properly by returning incorrect, excessive, or incomplete information.

Guidance Ideas:
An idea (not tested) is to include in the specification several examples of what the query should return. A dummy set of tables/records would have to be provided to demonstrate how the query might operate.

As visual query tools evolve, perhaps a compatible method of diagramming a query at the design stage can be proposed.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict □ Synergy	Project: KMM - Numerous Projects	Category: User I/F Layout	Date of Interview: 11/23/94 16:00 Frequency of Obsvervation: Usually
SE Application: □ Manual ☒ CASE □ N/A	SE Method: Any	Visual Tool: Any	

Description:
The UI Layout facilities provided by most every CASE tool are provide little or no benefit over paper or drawing tool sketches.

Circumstances:
Since CASE tools do not generate native code for any of the visual tools in common use, the UI must be reimplemented manually. (Some CASE tools do generate C source code that will give a look and feel like that of the design, but that source code is not usable by any of the typical visual tools, such as Visual Basic, PowerBuilder or SQLWindows.)

Guidance Ideas:
In general, do not use the CASE tool to flesh out your GUI design. Instead, use the visual programming tool itself to sketch it out, but keep the result grouped with the design package. Later, when implementation begins, the GUI file can be copied into the source area and used as a starting point for developing the application.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: □ Conflict ☒ Synergy	Project: KMM - Numerous Projects	Category: DB Schema	Date of Interview: 11/23/94 16:15 Frequency of Observation: Often
SE Application: □ Manual ☒ CASE □ N/A	SE Method: Gane & Sarson/ERD	Visual Tool: Visual Basic/PowerBuilder	

Description:
The structured data modeling features provided in today's CASE tools coupled with an automated schema generation facility provides significant productivity benefits.

Circumstances:
CASE tools typically provide an on-line data dictionary and the establishment of data domains. They also provide balancing routines which ensure that each layer of increasing detail has the same input and output structure as its parent. Automatic schema generators create SQL code in the Data Definition Language dialect of the chosen relational database product, typically covering the waterfront (e.g., Sybase, Oracle, Ingress, Informix, DB2, etc.). When the DDL source file is processed by the target database engine, the desired tables and columns are created with the proper type and size.

Guidance Ideas:
Structured data modeling should be applied whenever possible. One very successful combination used frequently by KMM is Gane & Sarson Data Flow Diagrams and Entity Relationship Diagrams. Automatic schema generation should be exploited whenever possible. Benefits of this approach include: an enhanced ability to create the initial prototype, enhanced ability to understand how objects are used (cross-referencing of which tables/columns are used where), enhanced ability to find things and discuss things.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: ☒ Conflict □ Synergy	Project: KMM - Numerous Database Projects	Category: Function Design	Date of Interview: 11/23/94 16:30 Frequency of Observation: Often
SE Application: □ Manual □ CASE ☒ N/A	SE Method: Gane & Sarson	Visual Tool: Visual Basic/PowerBuildar	

Description:
It is very difficult to describe desired functional behavior of modern GUI database front ends using the traditional process modeling paradigms offered in most software engineering methodolgies, Gane & Sarson being the case in point.

Circumstances:
The visual programming tools are anywhere from slightly to completely object-oriented in their design and usage, and all are event-based at least for their response to user interactions. Gane & Sarson, ERDs and the other more traditional modeling technques provide little support for expressing object-oriented or event-based behaviors. This leads to wordy narratives accompanying each diagram, or leaving the programmer to their own devices to translate a procedural design into its object-oriented, event-based counterpar.

Guidance Ideas:
Use DFDs and ERDs to model the data structures only (not behavior). Use Coad/Yourdon Event Models??? or State Transition Diagrams to model behavior.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type:
☒ Conflict
☐ Synergy

Project:
KBW - Projects with formal Distributions

Category:
Utility Modules

Date of Interview:
11/23/94 16:45

Frequency of Obsvervation:
Sometimes

SE Application:
☐ Manual
☐ CASE
☒ N/A

SE Method:
Any

Visual Tool:
Any component-based

Description:
It is very difficult to safely and reliably package a distribution diskette(s) when deploying an application built in a component-based tool or language (which includes most, if not all, of the visual ones). The problem includes both creation issues (e.g., knowing which components must be included) and installation/setup issues (e.g., version compatibility when two independantly created applications rely on the same component).

Circumstances:
The creation and use of Reusable Components is a key benefit of today's open and standardized software initiatives. Although still in its infancy, this goal provides a great deal of potential for leveraging the software development energies of the entire industry. It introduces a new set of challenges, however. Components include DLLs, VBXs, ODBC drivers and the like, usually supplied by the major software tool vendors, such as Microsoft, and their third-party developers.

Guidance Ideas:
Create a library of components which are clearly labeled and clearly defined/described.
Clearly specify which components are to be used in a project.
Do not put literal copies of a standard component in the config directory for a project, but rather point to it.
Insist that component vendors embed version and date in the component (to support programmatic version control prevention of overwriting a newer version with an older one).
Insist that component vendors provide forward compatibility for newer versions (so that if my application supplies a newer version of a component, someone else's application won't suddenly quit working).

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type:
☒ Conflict
☐ Synergy

Project:
RS - Customer Service Application

Category:
User I/F Layout

Date of Interview:
3/2/95 18:00

Frequency of Obsvervation:
Often

SE Application:
☐ Manual
☒ CASE
☐ N/A

SE Method:
SEER Methodology

Visual Tool:
SEER - HPS

Description:
Inherent behavior of Combo Box object only allowed the list box to contain static (predefined) choices; internal customer required dynamic additional to the list of choices. This lead to enormous expenditure of effort to provide the dynamic behavior (i.e., months, including hands-on support of tool vendor).

Circumstances:
Internal customer desired "commercial-grade" behavior of Combo Box (i.e., the purpose of a combo box is to allow the user to either pick from a list of previously entered choices or type in a new one). Most Windows applicaitons automatically add any newly typed-in items to the list for use by future users.

Guidance Ideas:
Cut losses early in the process. This could be accomplished by relaxing the requirements (in this case, by sacrificing the dynamic-update behavior). Alternatively, the development tool could be switched to one more congruous with customer requirements (probably only makes sense if there are numerous instances of mismatch, since most every tool will have some limitations).

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type:
☒ Conflict
□ Synergy

Project:
RS - Custmer Service Application

Category:
Function Design

Date of Interview:
3/2/95 10:15

Frequency of Observation:
Usually

SE Application:
□ Manual
☒ CASE
□ N/A

SE Method:
SEER Methodology

Visual Tool:
SEER - HPS

Description:
The functional design paradigm used by SEER did not land itself to "world class" look & feel and behavior specified by internal customer.

Circumstances:
Enormous time and energy expended trying to force fit desired functionality into tool not really designed to provide the desired look & feel or behavior.

Guidance Ideas:
Pilot a "good" initial deliverable based on the "flow" or inherent capabilities of the chosen development tools; then perform a value/cost assessment based on feedback on the pilot to determine whether the original requirements were indeed valid (==> switch tools for follow-on) or not (==> proceed with original tool to complete the final version).

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type:
☒ Conflict
□ Synergy

Project:
RS - Custmer Service Application

Category:
DB Schema

Date of Interview:
3/2/95 11:30

Frequency of Observation:
Often

SE Application:
□ Manual
☒ CASE
□ N/A

SE Method:
SEER Methodology

Visual Tool:
SEER - HPS

Description:
There is a "one-way" path from CASE model to schema; CASE also provides automatic normalization of data and automatic naming of tables and attributes. The auto-generated names are very cryptic in nature. Therefore, the developer typically edits the schema to make the names more "developer-friendly."

Circumstances:
Once the schema has been edited, the CASE representation is out of date, and must either be manually updated (i.e., all edits and maintenance of the schema must be done twice), or else the CASE representation must be abandoned.

Guidance Ideas:
Change to a tool that has reverse engineering capability (so that changes to the schema can be fed back into the model). Change to a tool that generates schema that are so good that they do not have to be edited; make maintenance changes in the CASE model and regenerate schema each time. Just use the CASE tool to develop the initial version and then abandon it.

Software Engineering Methods versus Visual Programming Tools/Languages
Synergy/Conflict Observations

Observation Type: □ Conflict ☒ Synergy	Project: RS - Customer Service Application	Category: Event-Based Design	Date of Interview: 3/2/95 18:45 Frequency of Observation: Often
SE Application: □ Manual ☒ CASE □ N/A	SE Method: SEER Methodology	Visual Tool: SEER - HPS	

Description:
The SEER product and method were developed with event-based, client/server computing application development in mind. Thus, the development process and implementation process for applications targeted for such an architecture are excellent.

Circumstances:
For the subject project, the SEER product and methodology were chosen because the target application was slated for running multiple data servers on multiple platforms (VSAM on mainframe, DB2 on mainframe, and DB2.2 on OS/2 PC), and taking advantage of event-based programming. These aspects of the environment matched up very well (the problems mentioned in the other observations not withstanding).

Guidance Ideas:
Match your tool's capabilities to the implementation architecture.

VITA

Mr. Touchton received a Bachelor of Science in Engineering (Nuclear Engineering) from the University of Florida in 1974 and a Master of Science in Nuclear Engineering from Carnegie-Mellon University in 1977. He is a licensed Professional Engineer in the state of Florida.

Mr. Touchton has 20 years experience including 10 years as a nuclear engineer for Westinghouse and a small consulting firm, and 10 years as a consultant in the application of advanced computing technologies for PathTech Software Solutions, Inc. Mr. Touchton is a co-founder and President of PathTech. His areas of expertise include the design and development of knowledge-based systems, client/server applications, and relational data bases. He is a champion of object-oriented programming and the incremental development paradigm.