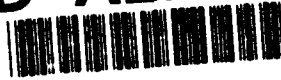


AD-A274 032



AFIT/GCS/ENG/93D-16

①

S **DTIC**
ELECTE
DEC 23 1993
A

**DEVELOPING A GRAPHICAL USER INTERFACE
TO SUPPORT A REAL-TIME
DIGITAL SIGNAL PROCESSING SYSTEM**

THESIS
Jeffrey C. Miller
Captain, USAF

AFIT/GCS/ENG/93D-16

Approved for public release; distribution unlimited

Developing a Graphical User Interface
to Support a Real-Time
Digital Signal Processing System

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Jeffrey C. Miller, B.S.E., M.B.A.
Captain, USAF

December 15, 1993

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Availability for Special
A-1	

DTIC QUALITY INSPECTED 3

Approved for public release; distribution unlimited

Preface

When I came to AFIT, I did not have a particular research area in mind. I knew that I wanted to further my knowledge and experience in both electrical and computer engineering. Although I did not make as much progress as I originally set out to make, working on the PSK system provided the breadth of experience I sought. I highly recommend further study of this system to anyone seeking an eclectic AFIT educational career.

Any substantial technical project is, almost by definition, a group endeavor. This thesis was no exception. I would like to express my gratitude to the members of my thesis committee for their advice, guidance, and support: to Maj Mark Mehalic, for overall direction and helping me find my way over, around, and through the rough spots; to Maj Paul Bailor, for asking the tough questions which caused me to reassess some of my assumptions and thinking processes; and to Capt Joe Sacchini, for keeping me straight on the essentials of signal processing.

Special thanks also go to Lt Col Tom Wailes for his review and comments on the material relating to the VMEbus, to Dr. Thomas Hartum for helping me break a mental block on aspects of object modeling, and to Col Robert Hanlon, former commander of the 1843rd Engineering Installation Group. His insistence on use of electronic mail started a long chain of events which eventually resulted in my application and admission to AFIT.

I'd also like to thank the members of the "morning coffee club" – Jay Cossentine, Vince Droddy, Warren Gool, and John Keller – for their special brand of humor and encouraging words over those first doses of "synapse lube" every morning.

Finally, much appreciation goes out to my wife, Pat; my daughter, Alicia; and my son, Marcus. Without their support, understanding, and tolerance of many late nights, delayed family outings, and stacks of technical material piled around the house, this thesis would not have been possible.

Jeffrey C. Miller

93 12 22 173 iii

93-31060


Table of Contents

	Page
List of Figures	x
List of Tables	xi
Abstract	xii
I. Introduction	1
1.1 Background	1
1.2 Problem Statement	4
1.3 Assumptions	5
1.4 Scope	5
1.5 Approach and Methodology	6
1.6 Thesis Overview	6
II. Literature Review	8
2.1 Introduction	8
2.2 Background	9
2.3 Uses of DSP Systems	10
2.4 New Software Tools and Interfaces	11
2.5 Real-Time DSP Architectures	13
2.6 Object-Oriented Development	14
2.6.1 Real-World Modeling	15
2.6.2 Increasing Levels of Detail	15
2.6.3 Support for Software Engineering Practices	16
2.7 Justification for Object-Oriented Development	16
2.8 User Interface Development	17

	Page
2.8.1 User Behavior Patterns	17
2.8.2 User Interaction Strategies	18
2.8.3 Empirical Guidelines	21
2.9 Reuse and Reengineering Strategies	22
2.10 Summary	24
 III. System and Requirements Analysis	 25
3.1 Introduction	25
3.2 Current System Hardware Configuration	25
3.2.1 The Ironics IV-3206 Single Board Computer	28
3.2.2 The Chrislin Industries CI-VME40 Dual-Ported VME Mem- ory Card	28
3.2.3 The Ironics IV-3272 Full Speed Data Transporter and IV- 3272-PIO Parallel Interface Daughter Card	29
3.2.4 The Vigra MMI-250 High Resolution VMEbus Graphics Controller Card	29
3.2.5 The Array Microsystems a66540 VME Frequency Domain Array Processor Card	30
3.2.6 The Buffer Memory	30
3.2.7 The SRL-Developed Analog-to-Digital Converter Card .	31
3.2.8 The SRL-Developed I/Q Splitter Card	31
3.2.9 The SRL-Developed Digital Filter Card	31
3.2.10 The SRL-Developed Digital Output Buffer Card	32
3.2.11 The SRL-Developed Dual Digital-to-Analog Converter Card	32
3.2.12 The Fujitsu Hard Drive	32
3.2.13 The Panasonic Floppy Drive	32
3.2.14 The Archive Scorpion Tape Drive	32
3.2.15 The NEC 5D Multimode Graphics Display	33
3.2.16 The DEC VT-340 Terminal	33

	Page
3.2.17 The Logitech Mouse and IBM PC/AT Keyboard	33
3.3 Current System Software Configuration	33
3.3.1 The OS-9/68000 Real-Time Operating System	34
3.3.2 The μ MACS Text Editor	34
3.3.3 The SYSDBG System State Debugger	34
3.3.4 The C Compiler	35
3.3.5 a66540 Support Library	35
3.3.6 IV-3272 Support Library	35
3.3.7 The RAVE Package	36
3.3.8 SRL-Provided Routines	37
3.4 The System Memory Map	37
3.5 The OS-9 Directory Structure	41
3.6 OS-9 System Configuration/Boot Sequence	46
3.7 Requirements Analysis	49
3.7.1 User Interface	49
3.7.2 Signal Processing Commands	49
3.7.3 System Commands	50
3.7.4 Operational Concept	51
3.8 Summary	51
IV. Software Analysis and Design	52
4.1 Introduction	52
4.2 Analysis Process and Models	52
4.2.1 Object Model	57
4.2.2 Dynamic Model	64
4.2.3 Functional Model	69
4.3 Design Process and Models	76
4.3.1 Analysis Objects Operations Addition	77

	Page
4.3.2 User Interaction Style Definition	81
4.3.3 Solution Space (Lower-Level) Object Addition	82
4.3.4 Algorithm and Data Structures Definition	91
4.3.5 Control Method Definition	92
4.3.6 Updating the Analysis Models	93
4.4 Summary	99
V. Software Implementation Plan	102
5.1 Introduction	102
5.2 Implementation Procedure	102
5.2.1 Translation into a Non-Object-Oriented Language . . .	103
5.2.2 Code Generation and Editing	103
5.2.3 User Interface Generation	104
5.2.4 Code Re-use	105
5.2.5 Code Packaging	105
5.2.6 File Management and Configuration Control	106
5.3 System Administration	106
5.4 Interface Prototyping	107
5.5 Summary	109
VI. Conclusion and Recommendations	110
6.1 Introduction	110
6.2 Summary	110
6.3 Recommendations	111
6.3.1 Buffer Memory	112
6.3.2 Software Implementation	112
6.3.3 System Development Environment	113
6.4 Lessons Learned	114
6.5 Conclusion	115

	Page
Appendix A. System Behavior Scenarios	116
Appendix B. Operation Algorithm Definitions	121
B.1 User Interface	121
B.2 Operating System Interface	123
B.3 File Interface	127
B.4 Real-Time Front End Processor Interface	132
B.5 Signal Processing Hardware Interface	134
B.6 IV-3272 Interface	135
B.7 a66540 Interface	136
B.8 Discrete Digital Signal	137
B.9 Signal Sample	137
B.10 Time-Domain Signal	137
B.11 Frequency-Domain Signal	137
B.12 Window	138
B.13 Menu	138
B.14 Choice Item	138
B.15 Data Input Box	139
Appendix C. Source Code Sources	140
C.1 User Interface	140
C.2 Operating System Interface	140
C.3 File Interface	140
C.4 Real-Time Front End Processor Interface	141
C.5 Signal Processing Hardware Interface	141
C.6 IV-3272 Interface	141
C.7 a66540 Interface	141
C.8 Discrete Digital Signal	141

	Page
C.9 Signal Sample	142
C.10 Time-Domain Signal	142
C.11 Frequency-Domain Signal	142
C.12 Window	142
C.13 Menu	142
C.14 Choice Item	143
C.15 Horizontal Bar	143
C.16 Pop Up Box	143
C.17 Data Input Box	143
Appendix D. System Operation	144
D.1 System Initialization, User Login, and User Creation	144
D.2 File Transfer	145
References	146
Vita	149

List of Figures

Figure	Page
1. PSK System Block Diagram	3
2. Detailed PSK System Architecture	27
3. System Object Model – Major Subsystems and Front End / DSP Hardware	54
4. System Object Model – General Purpose Computing Hardware	55
5. System Object Model – User Interface Software Objects	56
6. Software Analysis Object Model	58
7. Event Flow Model	68
8. Software Analysis Dynamic Model – User Interface State Diagram	70
9. Software Analysis Dynamic Model – Signal Processing Hardware State Diagram	71
10. Software Analysis Dynamic Model – Real Time Front End Processor State Diagram	71
11. Software Analysis Functional Model – Data Flow Diagram	74
12. User Interface Scheme	83
13. Software Design Object Model – Signal Processing and Interface Objects .	95
14. Software Design Object Model – User Interface Objects	96
15. Software Design Dynamic Model – User Interface Object	97
16. Software Design Dynamic Model – Real-Time Front End Processor Interface Object	98
17. Software Design Dynamic Model – IV-3272 and a66540 Interface Objects .	98
18. Software Design Dynamic Model – Horizontal Bar (Menu) Object	98
19. Software Design Dynamic Model – Pop-Up Box (Menu) Object	99
20. Software Design Dynamic Model – Data Fill-In (Menu) Object	99
21. Software Design Functional Model – Data Flow Diagram	100

List of Tables

Table		Page
1.	Global System Memory Map	38
2.	Example Scenario	66

Abstract

A graphical software user interface for a VMEbus-based real-time digital signal processing system was designed. User requirements were defined and the Rumbaugh object-oriented analysis and design technique was applied to analyze the requirements and produce an object-oriented design. The software design includes a graphical, mouse- and keyboard-driven user interface, specialized hardware driver modules, and operating system interfaces. An implementation plan was also developed to map the design into the C programming language using existing system code, automatically generated code, and newly written code. Based on the implementation plan, a limited software system prototype was successfully developed and demonstrated. The system can be used to analyze signals previously recorded on disk or sampled in real time. Analyzed signals can be displayed either as a set of discrete samples or as a graph in either the time or frequency domains. The system includes real-time sampling hardware, specialized DSP hardware, general-purpose computing hardware based on the Motorola 68030 microprocessor, mass storage media, and a high-resolution graphical display.

Developing a Graphical User Interface to Support a Real-Time Digital Signal Processing System

I. Introduction

1.1 Background

Signal processing, in its various forms, is an important part of both the electrical engineering and computer science fields. Over roughly the last decade, digital signal processing (DSP) has grown in importance as a useful method to analyze communications signals and other types of data streams; in particular, real-time DSP has evolved into a technology suitable for both signal processing and system monitoring and control. In the Air Force, there is interest in real-time DSP as a method for analysis of various electronic signals which might be encountered on the battlefield and as a method to respond to and counter those signals. Specifically, the National Air Intelligence Center (NAIC, formerly the Foreign Aerospace Science and Technology Center (FASTC)) conducts research in several areas of interest to the Air Force, including the application of DSP techniques to signal analysis. To that end, one of their projects was to contract for the development of a real-time, high-speed Versa Module Eurocard bus (VMEbus) based DSP system for research into new DSP algorithms, techniques, and applications. This system was delivered to them in 1990 by Systems Research Laboratories (SRL), Inc., under an open-ended tasking contract for research in the DSP area [1].

The system, known as the Phase-Shift Keying (PSK) Matched Filter system, is designed to analyze signal returns from a spread spectrum radar. The system designer made certain assumptions about the nature of the received signals, including their amplitude, frequency, phase, noise, and interference characteristics. These considerations drove the fundamental design of the system. It is designed to analyze analog input signals in the range 0 to 10 MHz which have been down-converted from an unspecified frequency range. As originally envisioned in the tasking contract, the analog input signal is intended to represent the returns from long-range, spread spectrum radar.

The system is divided into three major portions: the front end processor, the digital signal processors, and the central processing unit (including support hardware). The front end processor consists of the hardware required to sample and digitize the input analog signal. The input signal is applied to an anti-aliasing filter followed by a 20 MHz analog-to-digital (A-to-D) converter. Following the A-to-D converter are a splitter board to divide the sampled analog signal into in-phase (I) and quadrature (Q) components, a pair of low-pass filters to remove the mixing sum products from the output of the splitter board before the data is sent to the buffer memory, a pair of digital-to-analog conversion processors to provide analog I and Q outputs, and a pair of buffer memories to send the digital I and Q data on to the main processors.

The next portion of the system is the main digital signal processors. An Array Microsystems FFT card is used, which can accept the full 32 bits of I and Q data at the specified data rate. This board is the heart of the system, providing the desired analysis of the input signal and generating a direct digital output which can optionally be stored in memory or sent over to the graphics processor for display.

The final portion of the PSK system is the VMEbus controller and associated hardware. The system is based on a VMEbus computer system using a Motorola 68030 microprocessor. The main data path from the front end to the correlation processor does not travel over the main data path of the VMEbus. Instead, the real-time signal data is transferred over the user-defined portion of the VMEbus. The signal data can be moved over to the main VMEbus data path by an IV-3272 data transporter. This is done so that the 68030 can write the signal data to memory or send it to the graphics board for display on the color monitor (NEC 5D). The system also has mass storage peripherals, including a floppy disk drive, hard disk drive, and a tape backup device. An overall system block diagram showing the major components is shown in Figure 1.

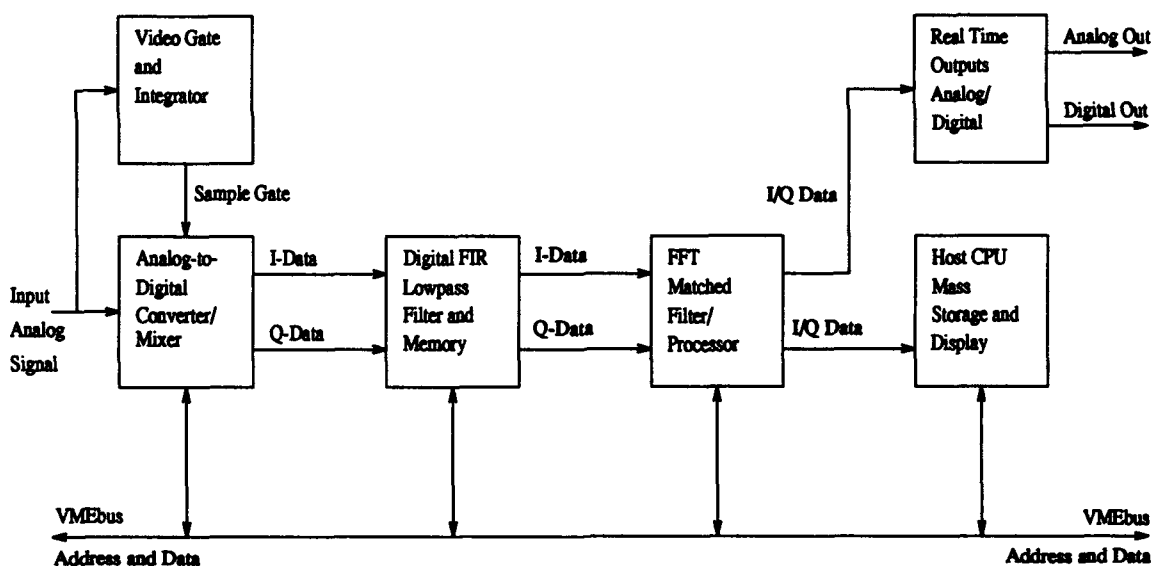


Figure 1. PSK System Block Diagram

Because the system is intended for eventual use as a real-time analysis tool, it operates under the Microware OS-9 real-time operating system, which is a multitasking operating system similar to UNIX. A variety of software development tools are loaded on the system,

including a text editor, debuggers, and a C (but not C++) compiler. Also included are driver routines for the various VME cards, a graphical interface development utility, and demonstration files from the SRL development effort. This software is currently stored on the hard disk drive in a tree directory structure, and forms the development environment that is used to develop the proposed user interface system. Chapter III provides greater detail on both the system hardware and software.

1.2 Problem Statement

The PSK system was delivered without a software user interface or overall control routines. As a result, the system must be manually initialized and configured for each use, including possibly the development, compilation, and execution of C source code. This startup process requires user knowledge of the fine details of OS-9 programming, the actual memory map of the system, and basic input/output routines which should be performed by the system. No menu, other well-defined user interface, or help routines are present, so the user must search the incomplete system documentation to determine how to operate the system. In essence, the system as currently configured is unusable for its intended purpose. This problem can be summarized as follows:

Using the best available software engineering techniques, design a software system which, when implemented, will make the PSK digital signal processing (DSP) system available for real-time DSP use. In this context, "make the ... system available" means that a user knowledgeable in DSP techniques and analysis, but not necessarily in computer systems, will be able to perform desired analyses without writing any new computer software.

This problem can be solved by finding solutions to the following objectives:

1. Design a series of software routines to allow a user expert in signal processing, but not computers, to activate the PSK system, select the type of analysis desired from possibly a menu or another predefined set of analyses, and allow presentation of results in a predefined format.
2. Design the support routines for the menu system to include applicable DSP algorithms, input/output routines, and file maintenance routines.

1.3 Assumptions

For this thesis, several assumptions about the DSP system were made. The system hardware, with the exception of the buffer memory between the DSP processors and the general-purpose computer, was assumed fully functional and working correctly. The basic software provided (the operating system, debuggers, editors, utilities, and compiler) were assumed to work correctly. This last assumption did not always turn out to be legitimate, as documented in Chapter III.

1.4 Scope

This thesis effort was limited to the design of the software system described in the objectives above. No attempt was made to design or write any sort of generic DSP software package; however, the software system should be designed to be easily modifiable for future enhancements. No development work with or upgrades to the system hardware were planned or accomplished. The overall effort is finished when the overall design is successfully completed.

1.5 Approach and Methodology

To accomplish the objectives set forth above, the following actions and activities were performed:

1. Thoroughly analyze the current system configuration and resolve apparent discrepancies in the supplied documentation. Determine the functions performed by the software as currently implemented. Recover the existing software design as much as possible. Develop a library of source code listings for the delivered software as written by SRL.
2. Perform a requirements analysis using object-oriented techniques. From that, determine the difference between what is required and what is currently supplied. From the results of that comparison, determine needed algorithms, library routines, and system configuration.
3. From these requirements, develop the software specification. This will be used as the baseline for comparison during testing.
4. Design the user interface and supporting library modules for algorithms and maintenance using object-oriented design.

1.6 Thesis Overview

This thesis is divided into six chapters. Chapter II, Literature Review, is a summary of current work being done with real-time DSP systems and outlines background information relating to object-oriented development and user interfaces. Chapter III, System and Requirements Analysis, is a detailed review of the PSK system as developed by SRL

and defines the requirements to be satisfied by the design. Chapter IV, Software Analysis and Design, is the actual analysis and design of the software system. Chapter V, Software Implementation Plan, outlines the steps which should be taken to develop the actual code for the system and related issues. Chapter VI, Conclusion and Recommendations, contains a summary of the work performed, recommendations for future work, and a description of lessons learned.

II. Literature Review

2.1 Introduction

This chapter surveys the current state of real-time DSP systems and describes the most important trends in the development and usage of those systems. It also outlines key software development concepts necessary for successful development, including object-oriented analysis/design and user interface design. The purpose of this review and discussion is to lay the system engineering framework for the software design effort to follow.

First, the chapter gives a brief background on the development of DSP systems and summarizes work previously done at AFIT in this area. Then it describes the current uses to which DSP systems are being put, followed by a description of new software tools and interfaces which are making DSP systems easier to design and use. It also describes new architectures that are enhancing the real-time capabilities of DSP systems.

Following the discussion of DSP systems, the chapter discusses various software engineering concepts necessary for a successful user interface design for the PSK system. First, key concepts of object-oriented analysis and design, including support for the notion of using object-oriented design in a real-time environment are discussed. The chapter also outlines current user interface development methodologies, including a brief outline of user behavior patterns, user interaction strategies, and empirical guidelines for the development of user interfaces. Finally, the chapter concludes with a discussion of software reuse strategies and how they apply to the user interface system being developed.

2.2 Background

Digital signal processing is a method to analyze and modify electrical signals produced by a variety of sources: a communications line, a radar return, or a sensor sensing any time-varying real world quantity (for example, a vibration sensor). As commonly implemented, this method involves converting the analog input signal into a digital format (known as sampling and quantization), transforming this time-domain digital signal into a frequency-domain representation using a Fourier transform algorithm (for example, one of the Fast Fourier Transforms), and performing calculations on and modifications to the resulting bit stream according to some predefined algorithm. The resulting digital output stream can be used in almost any way: it can be displayed on a graphical display for analysis, converted back to an analog form as part of a communications system, or used to provide real-time control of a system. The flexibility and power of DSP and DSP systems come from the ability to manipulate the bit stream at will using software and the specially designed DSP processing chips optimized for typical DSP calculations [2].

At AFIT, there have been three previous theses dealing with implementation of real-time DSP. In 1982, Todd developed a basic DSP processor using a Hewlett-Packard HP21MX general purpose computer [3]. His work concentrated on overall system development and integration and only implemented a limited subset of DSP tasks. His work was limited due to the primitive hardware and software available, as well as the difficulty of building a new system essentially from scratch. As a follow-on, in 1983, Bengtson implemented a software interface and DSP routines on the same HP21MX general purpose computer [4]. This system successfully implemented data generation, Fast Fourier Transform

(FFT), Inverse FFT, and graphics display, using Weinberg's Structured Analysis methodology. This software development methodology was one of several similar methodologies popular at the time, all of which revolved around structured functional decomposition of a software system to be developed. Hardware limitations prevented implementation of other planned functions, including real-time data acquisition, analog-to-digital (A/D) conversion, and processing. This system did successfully implement a user interface to hide programming and operating system details from the user. Notable in both Todd's and Bengtson's efforts was the absence of a dedicated hardware DSP data path and processors in their system architecture.

Mills implemented DSP routines and a limited software interface for the Modulation Detection and Classification receiver system developed by ITT [5]. Like the system under study in this thesis, Mills' system was delivered from the contractor without a fully-functioning interface; however, his effort concentrated primarily on developing efficient FFT and data transfer code to meet the real-time constraints imposed by system requirements. Both Mills' system and the current system under study have common features: the presence of specialized DSP hardware and the requirement to process incoming data in real-time. For these reasons, techniques used and recommendations made by Mills are considered useful in avoiding problems in the current study.

2.3 Uses of DSP Systems

Traditionally, DSP systems have been applied to filtering of communications signals, enabling easy implementation of filters that would otherwise be difficult to implement with strictly analog hardware. Now, however, there are a growing number of applications using

DSP systems outside of the communications area. These applications have come about due to the increasing performance of DSP hardware and, to a lesser extent, software [6].

Tesmer [7] outlines two new uses of DSP systems made possible by the development of higher performance hardware and the evolution of a workable real-time architecture (discussed later). The first application is active vibration cancellation in a space-based telescope. In this application, the DSP system uses the inputs from vibration sensors to generate control signals to drive motors which compensate for the vibration. The second application simulates real-time acoustic environments. In this application, the DSP system generates signals to which a system under test must respond; the DSP system can be reprogrammed to generate test signals which can be used to simulate a variety of environments. In these applications, the DSP system provides an easily programmable, predictable control system not available from a purely analog implementation. Ease of programming, however, is itself an area of important work and is discussed next.

2.4 New Software Tools and Interfaces

Software has historically been a limiting factor in the development of DSP systems. Writing software has been a labor-intensive process since compilers for high level languages, such as C, have not generally been available until recently for the specialized DSP processors used in high-performance DSP systems [8]. Instead, the DSP designer has been forced to program entirely in the native assembly code of the DSP processor used. This use of a low level language substantially slows the development process, even though some rudimentary development tools such as editors, debuggers, and linkers are available.

According to Mather [9], user interfaces and tools for DSP systems are improving. C compilers and graphical interfaces are starting to be developed for the DSP processors now on the market. In some cases, commonly-used DSP algorithms are provided with these tools so developers need not rewrite existing algorithms and can spend more time on system development. These tools are available for a variety of development platforms, including IBM-compatible PCs, Apple MacIntoshes, and Sun and similar workstations.

There is disagreement, though, on the efficiency and effectiveness of these tools. Robinson [8] gives two reasons why DSP system designers are still using assembly-level programming:

- The higher level tools do not incorporate necessary control interfaces or mechanisms.
- Code produced by the higher level tools tends to be inefficient compared with directly (hand) generated assembly code.

Robinson believes, though, that these tools are still in the early stages of development and that their evolution will follow that of microprocessor development tools. If the efficiency and effectiveness of higher-level development tools and the code they generate increase over time, DSP system performance, ease of use, and ease of design will also continue to improve.

One DSP research and development tool available at AFIT is the Khoros system. This system combines signal and image processing applications with an underlying graphical user interface and visual programming system [10]. The signal and image processing applications currently included with the Khoros system include the most common algorithms used in DSP: Discrete and Inverse Fast Fourier Transforms, Spectral Estimation,

Finite and Infinite Impulse Response filter designs implementing lowpass, highpass, bandpass, and bandstop filters, and Discrete Convolution. However, there is more to the Khoros system than just signal processing. It is designed to be a general purpose graphical user interface development system and thus contains automatic application interface generators. The Cantata Visual Language interface to Khoros extends the interfaces generated by the Khoros system to include glyphs, or icons, which can be connected to form a desired signal or image processing system which can then be studied and used to process actual or simulated data. The glyphs in Cantata represent underlying processing algorithms applied to signals and provide a graphical technique for building (simulated, non-real-time) signal and image processing systems.

The automatic user interface code generation capability of Khoros is similar to, but more powerful than, the capability provided by the Real-Time Audio/Video Environment (RAVE) package on the PSK system which will be discussed in Chapter III. Many of the same user interaction concepts are used in both Khoros and RAVE, however. The design of the PSK user interface will use not only those concepts, but the user interaction concepts and strategies to be discussed later in this chapter.

2.5 Real-Time DSP Architectures

As noted previously, improvements in DSP system performance have paved the way for development of new applications. One of these applications is the development of DSP systems to provide real-time control of other systems. The architecture that has evolved for this application is the combination of a dedicated DSP processor with another host processor to handle user interface and other housekeeping tasks.

This architecture evolved due to the conflict between the need to immediately service incoming data in a real-time system and the interrupt-driven nature of a general purpose processor. A DSP processor is optimized for particular types of calculations on an incoming data stream, but does not handle random user interrupts well. A general-purpose processor, on the other hand, handles user interrupts well, but this property makes it unsuitable to handle incoming data streams which can and will not wait for interrupts to be handled. Teaming these two types of processors in a system provides the advantages of both, with the disadvantages of neither, provided that communication between the two is handled correctly [7].

A similar architecture has been used for real-time data collection on board a laboratory ship [11]. In this case, a real-time data management system that recorded and processed acoustic data was linked to Sun workstations which provided user interface to the system and the ability to look at the incoming data in near real time.

Another architecture development is the inclusion of multiple DSP processing chips on a single card [6]. This architecture can provide several real-time processing capabilities: processing of multiple data streams (with each stream being independently processed), processing of single data streams which are too fast for a single DSP chip to process, or even processing of multiple fast data streams. When combined with a host processor as outlined above, this architecture forms an extremely powerful, albeit expensive, system.

2.6 Object-Oriented Development

Selection of a development technique for this system is a crucial decision, as it affects the efficiency and accuracy of the development process and the overall effectiveness and

quality of the finished system. This thesis effort uses Rumbaugh's Object-Oriented (OO) development methodology [12]. This technique offers a different perspective on problems compared to the functional decomposition techniques used for previous development of similar systems. The OO perspective is summarized in the following paragraphs.

2.6.1 Real-World Modeling. The OO technique models the real world as consisting of *objects*, which are representations of the important properties of real world entities. These properties are modeled as *attributes*, which describe the entity, and *operations*, which describe functions done by or to the entity. Only the salient characteristics of real world entities are abstracted into the object which represents the entity. Furthermore, objects can be related by associations which model real-world relationships between the objects, or they can be related hierarchically so that an object inherits attributes and operations from another object. For example, an engine object (with its attributes and operations) may be part of an automobile object (with its own attributes and operations). In this case, the engine may also inherit attributes from the automobile object, such as a serial number.

2.6.2 Increasing Levels of Detail. The OO technique can be used from the initial analysis phase of a project through its implementation. During analysis, the object model closely parallels the real-world entities being modeled. As the model is moved into design, details are added to reflect the overall system design strategies. Finally, during implementation, the object model is transformed into the desired programming language by the addition of specific implementation details, while still maintaining the correspondence to real world entities.

2.6.3 Support for Software Engineering Practices. The OO strategy supports good software engineering practices such as data abstraction, information hiding, and reuse [12]. Data abstraction is the process and method by which the important properties of the data are highlighted and retained while the unimportant properties are ignored. Information hiding is the philosophy that a software module shares only essential information with other program or system modules, and that operations on module data are performed only by tested, approved procedures. Reuse is the concept that code should be made generalizable enough to be useful across an application domain, so that new systems do not have to be written from scratch, but can be modifications of existing ones, compositions of previously existing code modules, or parameterizations of generic code modules. The manner in which objects are constructed supports these concepts. An object's attributes are a form of abstraction and information hiding; only the necessary and essential information about the object and its properties is stored. The object's operations support information hiding; all other objects may only interact with a particular object using the public interface provided by the object's operations, with the implementation details of those operations hidden inside the object. Finally, well-constructed objects (i.e., constructed so that they apply across multiple application domains) can be reused in a variety of systems.

2.7 Justification for Object-Oriented Development

The OO development method is the most appropriate for the PSK system for several reasons. First, because of the combination of attributes and operations within objects, it more closely models the real world than the functional decomposition development methods used in the past [12]. Second, it will allow the encapsulation of existing software on the

PSK system, thus promoting reuse. Third, it is adaptable to real-time design by virtue of adding timing requirements to the various objects [13]. Finally, it is the technique in widest use within the AFIT software engineering curriculum.

2.8 User Interface Development

Development of a logical, powerful, and easy to use interface is one of the keys to a successful implementation. The study of user interface design, also known as human computer interaction, is a substantial field. For the purposes of this investigation, well-established principles and guidelines are used; the user interface designed in this study will reflect conformance with the strategies and guidelines in the published literature.

2.8.1 User Behavior Patterns. Shneiderman has developed three levels of users and usage profiles [14]. He divides all users' knowledge into two categories: syntactic, which deals with the arbitrary, system-dependent details of actually using the system; and semantic, which deals with the concepts and operations within a given knowledge area. Shneiderman's three levels of users are novice, knowledgeable intermittent, and frequent users:

- Novice users have little, if any, syntactic knowledge about the system. They also may have little semantic knowledge of computer systems and the application or task area in question. This group needs the highest level of interface-based assistance, on-line help, and informative feedback from the system.
- Knowledgeable Intermittent users have greater semantic knowledge of the task and the computer system, but do not necessarily recall the syntactic knowledge necessary

for successful system use. This group needs less assistance from the system, but can still benefit from informative feedback to reinforce their recall of system details.

- Frequent users have a high level of semantic and syntactic knowledge, and do not require much assistance from the system in order to use it effectively. In fact, this group would prefer shortcuts and other tools to increase the speed with which they use the system.

Clearly, designing an effective user interface that will meet the needs of all three groups is a challenge. Shneiderman proposes a layered strategy [15]: allow novice users a minimum set of commands and tasks, with user control over how many tasks are presented; knowledgeable intermittent users will be allowed to modify the presentation details as desired; and frequent users will be presented with a minimum of detail.

As noted in Chapter I, this thesis assumes that system users will be knowledgeable about signal processing, but not necessarily about the PSK system in particular. This places most users initially in either the novice or knowledgeable intermittent user groupings. It is reasonable to expect, however, that most users will become knowledgeable over time, so allowances must be made for adaptation of the user interface to meet their needs. Examination of user interaction strategies will help resolve some of this difficulty.

2.8.2 User Interaction Strategies. A user interaction strategy or style is the set of objects and techniques which define how the user communicates with the system [15]. Shneiderman defines five of these styles [14]:

- **Menu Selection.** Users choose an action from a list of items and direct its execution.

This style is most helpful to novice and intermittent users, who appreciate the inher-

ent on-line help provided by the menu. The primary disadvantage of menus is the need to completely specify all possible tasks to be performed by the system when the menu is created.

- **Form Fill-in.** A predefined form is displayed on the screen and appropriate entries are made in the supplied blanks. This style is most appropriate for intermittent and frequent users, where data entry is required. Novice users may be confused by the semantic requirements, especially if task knowledge is low.
- **Command Language.** The user issues the desired command directly to the system in the system's native language. This style is best used by frequent users due to the semantic and syntax requirements; it can express complex commands in minimum space, however.
- **Natural Language.** The user issues the desired command directly to the system in the user's natural language (for example, English). This style is useful (for now) only for novice users in areas with a limited task domain.
- **Direct Manipulation.** The user points at pictorial representations (icons) of the objects of interest and moves them over other icons representing desired actions. This style is most useful for novice users, but can be useful for other users if designed and implemented to act quickly.

Shneiderman adds that a combination of these styles may be appropriate for systems where a range of users is expected. Hix and Hartson expand on these user interaction styles, especially in the area of menus [15]. They specify various kinds of menus, including the following:

- **Push-Button Menu.** A menu presenting multiple choices, usually with a default choice. These menus are generally used for semi-permanent display of alternatives.
- **Radio-Button Menu.** A menu presenting multiple mutually-exclusive choices. These menus are generally used when only one choice out of many can be made.
- **Check-Button Menu.** A menu presenting multiple non-mutually-exclusive choices. These menus are generally used when one or more choices out of many can be made,
- **Pull-Down Menu.** A menu whose title is always visible at the top of the screen, and can be summoned by selecting the title.
- **Pop-Up Menu.** A menu which appears at the current cursor location when a certain action is executed.

Hix and Hartson also classify windows and boxes as user interaction styles. Window objects are methods by which various tasks can be separated into defined areas on the display screen [15]. Box objects are used for shorter-term interaction with the system, such as system messages or text entry. Boxes can be thought of as a type of window created by or as part of a task running in a window.

Based on the information just presented, the most logical interaction style for the PSK system is the following:

1. Menus should be provided for the routine functions of the system, but command-line shortcuts should be allowed for expert users.
2. System-level displays and interaction should be kept away from the data display by using separate boxes and windows.

3. Data entry should be accomplished through use of fill-in forms to ensure that all necessary data is obtained. These fill-in forms also provide memory-joggers for novice and intermittent users.

2.8.3 Empirical Guidelines. Shneiderman provides eight "Golden Rules" of interface design [14]:

1. Strive for consistency: similar situations should require similar actions by the user, and identical terminology should be used for prompts, menus, and help screens.
2. Enable frequent users to use shortcuts: abbreviations, macros, and special keys should be available to speed interaction if desired.
3. Offer informative feedback: every user action should generate feedback, the extent of which should depend on the magnitude of the action's effect.
4. Design dialogs to yield closure: action sequences should have a start, middle and stop, and allow the user to disregard the finished action and prepare for the next.
5. Offer simple error handling: design the system to prevent serious errors; if prevention is not possible, then ensure that a method is provided to back out gracefully from the error and provide meaningful recovery instructions.
6. Permit easy reversal of actions: as many actions as possible should be reversible, to allow users to perform actions without fear of irreversibly damaging data.
7. Support internal locus of control: users want to feel in control of the system; they should be the initiators of actions, not responders to system actions.

8. Reduce short-term memory load: limited user short-term memory means that displays should be uncluttered, command sequences be kept short, and on-line help be available.

Hix and Hartson amplify these basic rules and add more specific rules to follow [15]. They also add that these guidelines may conflict, which requires designer interpretation to provide the best possible interface. Possibly the most detailed and comprehensive set of guidelines now available for the design of user interfaces is a report published by the MITRE corporation, containing 944 specific guidelines [16]. While too lengthy to summarize here, selected guidelines from the MITRE report were used along with the Shneiderman rules and the Hix/Hartson rules to guide the development of the PSK system user interface.

2.9 Reuse and Reengineering Strategies

Reuse was mentioned previously as a "good software engineering" practice which the OO development methodology supports. There are different levels of software reuse, however. Biggerstaff and Richter identify a primary issue of code reuse versus design reuse [17]. They contend that the highest payoff occurs not from code reuse, but from design reuse. They go on to say, however, that design reuse presents its own problems of how the design information is represented so that it can be reused. They also argue that code reuse is successful within limited application domains that are well-understood. Fortunately, the digital signal processing domain is just such a domain. It is limited and the mathematical basis is well understood. However, DSP hardware is rapidly changing, a condition which Biggerstaff and Richter argue is not conducive to code reuse.

Based on Biggerstaff's and Richter's criteria, code reuse is viable for the PSK system. The particular system hardware configuration is fixed and the driver software for the hardware components is already available. The challenge is to design the user interface and related components so that the design information is captured for future reuse and modification. Furthermore, the RAVE graphical user interface code generator has already captured user interface design information in the form of the predefined objects. These predefined objects are stored in the RAVE system which is used for interface composition. This information is thus available for future designers and implementors. For the lower level hardware dependent interface code, the user interface system design will capture the design information from the hardware modules. This information will be used during the Object-Oriented Design to create objects which are reusable in similar systems and applications by the nature of the OO design process.

The process of capturing the design of the current hardware and software is one element of the overall system engineering and re-engineering process necessary to develop the user interface for the PSK system. Chikofsky and Cross discuss the various steps and processes used in modifying an existing system [18]. Specifically, they identify *re-engineering* as the process which includes both *reverse engineering* and *forward engineering*. Reverse engineering, in turn, consists of *redocumentation*, the process of recovering documentation about the system which is for whatever reason not available, and *design recovery*, the process of combining available system information with domain knowledge to create an abstract description of the system to include what, why, and how it does the functions it does. The PSK system required nearly a full-scale reengineering process to recover all available design information, and resulted in the identification of source code which can be

reused either as-is or with modification. This information is presented as a part of Chapters III and V. The forward engineering of the PSK user interface system is presented in Chapters IV and V.

2.10 Summary

This chapter has reviewed the current status of DSP systems, especially as it applies to real-time systems. It gave a brief background of DSP, including previous AFIT thesis work, how DSP is used, software tools and interfaces now available and under development for DSP systems, and common architectures of real-time DSP systems. Unfortunately, there does not appear to be much academic research devoted to this topic, as shown by the relative lack of citations from the scientific literature. The commercial world, however, apparently is devoting a great deal of work to DSP systems.

The chapter also reviewed fundamental concepts of object-oriented analysis and design and showed why it is the most appropriate software development method for this thesis study. Finally, it reviewed basic user interface design concepts and highlighted an appropriate user interaction paradigm for this project. Based on the information found in the current literature, a system engineering perspective, incorporating techniques and knowledge from several domains, will be required to successfully design a useful software interface for the PSK system.

III. System and Requirements Analysis

3.1 Introduction

This chapter reviews the initial status of the hardware and software as it was provided by SRL and develops a requirements specification for the new system. It first outlines the functions and capabilities of each of the boards and modules in the PSK system, with special emphasis on the real-time DSP data flow and processing capabilities. Next, it outlines the software modules provided and assess their functionality. After that, it discusses three concepts crucial to understanding of the system: the system memory map, the OS-9 directory structure, and the OS-9 system configuration/bootup sequence. Finally, it develops the requirements specification for the system.

3.2 Current System Hardware Configuration

In order to fully understand the PSK system, it is necessary to understand the hardware modules and subsystems which make it up. The system currently consists of the following hardware items:

1. A rack-mount chassis containing a 20-bay VME card cage, power supply, and external connectors.
2. A separate chassis containing a Fujitsu 68 megabyte (MB) hard disk drive, a Panasonic high density floppy drive, an Archive Scorpion 45/60 MB backup tape drive, a SCSI controller, and a power supply.
3. A NEC 5D multimode high-resolution graphics display.
4. A DEC VT-340 terminal.

5. A Logitech three button mouse and IBM PC keyboard.

The VME card cage contains the following double-height (form factor 6U) VME cards, which comprise the heart of the PSK system. From left to right as they are installed in the system, they are:

1. An Ironics IV-3206 Single Board Computer.
2. A Chrislin Industries CI-VME40 Dual-Ported VME memory card.
3. An Ironics IV-3272 Full Speed Data Transporter, with IV-3272-PIO Parallel Interface Output daughter card.
4. A Vigra MMI-250 High Resolution VMEbus Graphics Controller card.
5. An Array Microsystems a66540 VME Frequency Domain array Processor card.
6. Two cards comprising the buffer memory, constructed under an EENG 699 special study [19].
7. An SRL-developed Digital Output Buffer card.
8. An SRL-developed Dual Digital-to-Analog converter card.
9. An SRL-developed Digital Filter buffer card.
10. An SRL-developed I/Q Splitter card.
11. An SRL-developed Analog-to-Digital Converter card.

The overall system architecture is shown in Figure 2. Now that the hardware modules have been identified, the following sections will elaborate on each of the system modules.

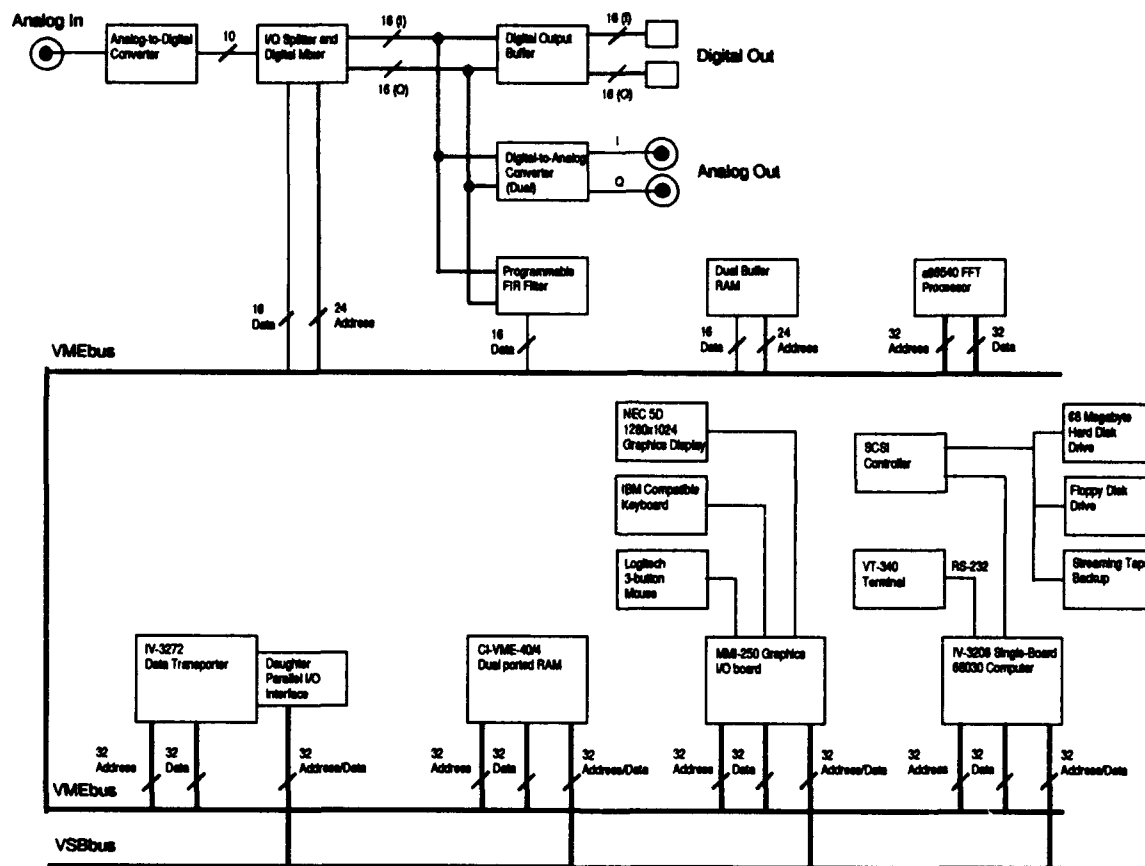


Figure 2. Detailed PSK System Architecture

3.2.1 The Ironics IV-3206 Single Board Computer. This card is the heart of the PSK system. Built around a Motorola 68030 microprocessor and 68882 co-processor running at 50 MHz, this board serves as the system controller and hosts the user interface software [20]. The board is provisioned with 4 MB of random access memory (RAM) and 128 kilobytes (KB) of read-only memory (ROM) containing the system boot monitor program. The RAM is accessible from both the VMEbus and the VME Subsystem Bus (VSB), as well as being accessible locally to the 68030. The base address of the RAM is set by on-board jumpers, and must match the RAM configuration loaded by OS-9 during bootup. The board is also equipped with a VSB controller and a Small Computer System Interface (SCSI) controller. The VSB controller oversees the operation of the VSB, which in this system is also used for transfer of real-time DSP data. The SCSI controller is the interface to the hard disk, floppy disk, and backup tape drive. Finally, the IV-3206 has four RS-232 compatible serial ports for asynchronous communication with terminals (including the system VT-340), and a parallel port which can be adapted for use with a parallel printer. The IV-3206 supports all of the permissible VMEbus data transfer addressing modes and sizes.

3.2.2 The Chrislin Industries CI-VME40 Dual-Ported VME Memory Card. The CI-VME40 card provides 4 MB of RAM directly accessible from either the VME bus or the VSB [21]. As such, this memory can be used as a buffer between the general-purpose and DSP processing portions of the system, or as intermediate storage by either portion. For example, test data could be read from the (relatively slow) hard disk and stored in this memory for high-speed access by the DSP hardware. The base address of this card

in both the VMEbus and VSB address spaces is also set by on-board jumpers and must match the bootup configuration loaded by OS-9.

3.2.3 The Ironics IV-3272 Full Speed Data Transporter and IV-3272-PIO Parallel Interface Daughter Card. This card combination is designed to move data between locations on the VME bus [22]. Containing a Texas Instruments TMS320C25 DSP chip, the IV-3272 can also perform DSP functions on the data it is transferring. However, the TMS320C25 must be separately programmed in its native assembly language to perform any DSP routines; otherwise, it has no effect on the transferred data. The IV-3272 essentially acts as a buffer between higher and lower speed devices on the VMEbus. The IV-3272 appears in the short address space of the VMEbus for configuration purposes; the exact addresses are also set by on-board jumpers. The parallel interface daughter card provides the interface to the VSB.

3.2.4 The Vigra MMI-250 High Resolution VMEbus Graphics Controller Card. This board provides I/O services, including graphics display, two serial I/O ports, keyboard mapping and translation, and stereo audio output [23]. All these facilities will be used with the exception of the audio output. The MMI-250 interfaces with both the VMEbus and the VSB, and can transfer data to and from both buses. The MMI-250 contains 4 MB of RAM to buffer image data, which is necessary due to the slower access times of the RAM relative to other devices in the system. This RAM, like the RAM on the IV-3206 and CI-VME-40 cards, is also addressable from both the VMEbus and the VSB. Its base address is also set by on-board jumpers and must match the base addresses loaded by OS-9 during bootup. The video portion of the MMI can support image sizes of up to 1280x1024

pixels, with simultaneous displays of either 2, 4, 16, or 256 colors. The serial ports can be programmed for asynchronous speeds up to 38.4 kilobits/s, and the keyboard port can translate scan codes of either IBM PC XT or AT keyboards. In the PSK system, one of the serial ports interfaces with a three button Logitech mouse. This mouse is used for interaction with the various display menus and controls to be developed.

3.2.5 The Array Microsystems a66540 VME Frequency Domain Array Processor Card. This board is the key to the DSP processing stream. It contains a digital array signal processor (DASP) which is optimized to perform radix-2 and radix-4 FFT operations [24]. Another processor controls the DASP and the on-board RAM. Seven banks of RAM are located on this board. Four banks are between the DASP and the VMEbus and VSB, and serve to buffer incoming/outgoing data. Two banks buffer between the DASP and the Controller, and one bank holds the user defined algorithms. The RAM can store data arrays up to 64K complex words in size, and is set up as a First In First Out (FIFO) buffer so that the board can simultaneously transfer data and perform an FFT. Both the control registers and the RAM appear in the VMEbus address spaces at a base address set by on-board jumpers; however, these addresses are not known to OS-9 at bootup since they do not represent mass storage available to OS-9 for its own use.

3.2.6 The Buffer Memory. These boards were constructed under an EENG 699 special study and were intended to complete the DSP data path, which was left incomplete at the end of the SRL effort. These cards accept the 10 high-order bits of the 16 bit I and Q signals from the I/Q card and relays them to the digital filter buffer card, using a "ping-

pong" technique: while one memory bank accepts input, the other provides output [19]. As currently constructed, the card is incomplete. It is missing one bank of memory.

3.2.7 The SRL-Developed Analog-to-Digital Converter Card. This card is the first card in the real-time DSP signal path. Incoming analog signals are sampled and converted to a 10-bit digital data stream by an Analog Devices AD9060 analog to digital converter [25]. The sampling rate and cutoff frequency are software configurable and are loaded into registers with addresses on the VMEbus. The register addresses are set using on-board switches and jumpers.

3.2.8 The SRL-Developed I/Q Splitter Card. This card is the second card in the real-time DSP signal path. The 10 bit words from the A/D card are split, using a Plessey PDSP16350 I/Q Splitter and Mixer chip, into the in-phase (I) and quadrature (Q) digital signals. Both the I and Q signals are 16 bit words and are routed to the VSB [25]. However, the VSB is used in a non-standard way but is isolated from the portion of the system which uses the VSB in the standard fashion [26].

3.2.9 The SRL-Developed Digital Filter Card. This card is the third card in the real-time DSP signal path. The 16 bit I and Q signals are each processed by four Plessey PDSP16256 Programmable FIR Filter integrated circuits (ICs) operating in cascade [25]. This filter is designed to operate as a low-pass filter to remove any aliasing products generated by the mixing/splitting process. Once filtered, the I and Q signals are returned to the VSB for access by the output buffer and graphics cards.

3.2.10 The SRL-Developed Digital Output Buffer Card. This card reads the digital I and Q signals from the VSB, buffers them, and makes them available for use on the external rear connectors [25].

3.2.11 The SRL-Developed Dual Digital-to-Analog Converter Card. Like the Digital Output Buffer card, this card reads the digital I and Q signals from the VSB, buffers them, converts them to analog form, and makes them available on BNC connectors located on the system rear panel [25]. Both the Digital-to-Analog converter card and Digital Output Buffer card read the data prior to, rather than after, the Digital Filter card. Because of this configuration, the data are corrupted with aliasing products and are currently of limited usefulness.

3.2.12 The Fujitsu Hard Drive. This device is located in the outboard peripheral chassis. It has a total formatted capacity of 68 MB, of which currently only nine megabytes is used. This disk contains all the known software for system, including vendor-supplied and user-developed software.

3.2.13 The Panasonic Floppy Drive. This device is also located in the outboard peripheral chassis. It is used primarily for loading new software into the system, other than software developed on the system itself. It can read and write to both OS-9 format and (through use of the PCF system utility) IBM-compatible MS-DOS format disks, both high and low-density.

3.2.14 The Archive Scorpion Tape Drive. Like the two disk drives, this device is located in the outboard peripheral chassis. Depending on which of two types of tapes

are loaded, this drive has a capacity of either 45 or 60 MB. As originally configured, the system could not access the tape drive.

3.2.15 The NEC 5D Multimode Graphics Display. This device is a high-resolution graphics display capable of emulating IBM VGA/MCGA display modes, MacIntosh display mode, and 800x600/1024x768/1280x1024 pixel display modes [27]. This display is intended to serve as the primary user interface device, and is supported by the MMI250 graphics card.

3.2.16 The DEC VT-340 Terminal. This device is used for the system console function, including system initialization and user interaction. Once the DSP application is started, this terminal is used for display of system error messages.

3.2.17 The Logitech Mouse and IBM PC/AT Keyboard. These items are used for interaction with the application software and are serviced by the MMI250 Graphics Board.

3.3 Current System Software Configuration

As with the hardware, to understand the PSK system it is necessary to be familiar with the software on the system. SRL supplied some basic software routines with the system, along with vendor-supplied software for some of the hardware items. These packages include:

1. The OS-9/68000 real-time operating system, including the μ MACS text editor, the SYSDBG system state debugger, and a C compiler.

2. FFT routines for the Array Microsystems FFT board.
3. Support routines for the Ironics IV-3272 board.
4. The Real-Time Audio/Video Environment (RAVE) package, which includes a graphics support editor and library, a presentation editor, and a file interchange library.
5. Various undocumented SRL developed test and demonstration routines.

These software packages are described in further detail as follows:

3.3.1 The OS-9/68000 Real-Time Operating System. The OS-9 operating system is similar to UNIX [28, 29]. It uses a similar hierarchical file structure to allow convenient separation of software packages. Many of the commands are similar, if not exactly the same. It allows multitasking and multiple users on the same system, a feature which will be essential for development of the PSK system. It differs from UNIX in that data and execution directories can be separately specified (in other words, they do not have to be the same directory); it allows definition of blocks of memory to be used only for specific purposes (known as *colored memory*); and explicit support for real-time processes by use of pre-emptive scheduling.

3.3.2 The μ MACS Text Editor. The μ MACS text editor is similar to the `emacs` text editor. It is screen and buffer oriented and uses a large subset of the `emacs` commands [30].

3.3.3 The SYSDBG System State Debugger. The SYSDBG system state debugger is a system-level debugging utility designed to give the programmer the ability to view and modify system memory and registers as necessary [31].

3.3.4 The C Compiler. The C compiler implements the full C language as defined by Kernighan and Ritchie [32]. A set of libraries is supplied, including user trap and I/O handlers, signal handlers, and math functions [33]. The C compiler consists of multiple parts: the executive, which handles the overall compilation process; the C preprocessor, which prepares the source code for true compilation by macro expansion and inclusion of other source files; the compiler itself, which translates the C source to 68020 assembly code; the optimizer, which (as its name suggests) attempts to streamline and optimize the assembly code produced by the compiler; an assembler, to translate the 68020 assembly code into binary object modules; and a linker, to link the object modules into an executable file. The assembler and linker can also be separately invoked on handwritten assembly code modules to produce executable code.

3.3.5 a66540 Support Library. The a66540 FFT Board was provided with a support library of C routines [24]. In all, 19 C language routines were provided to accomplish the following functions: initialize and reset the a66540; perform diagnostic tests; read program and data memories; write to program and data memories; and read/write control registers.

3.3.6 IV-3272 Support Library. The IV-3272 Support Library was supplied by Ironics in hardcopy format only. The code provided performs the following functions: set addresses; read and write registers; start and stop transfers; and initialize Transfer Parameter Blocks (TPB), which are data structures used to control the IV-3272. However, not all of the Ironics supplied code was loaded on to the system. Some of the code was

modified by SRL and placed in the IV3272 directory, where it was compiled and partially tested.

3.3.7 The RAVE Package. The RAVE package provides full support for user interface development and audio/video input and output. It includes four separate sub-packages: the graphics support library, the graphics file manager, a presentation editor, and the file interchange library. The packages are described as follows:

3.3.7.1 The RAVE Package Graphics Support Library. The graphics support library (GSL) is the interface between the RAVE-developed user interface application and the OS-9 operating system [34]. It uses the primitive video and audio tools and drivers provided by the graphics file manager to construct user interface objects, which can be either requests, controls, or indicators. Requests are menus used for user input; controls are user inputs configured as switches; and indicators are objects which appear as output devices, such as meters, dials or strip charts.

3.3.7.2 The RAVE Package Graphics File Manager. The graphics file manager (GFM) is a lower-level set of routines and drivers used for video and audio input/output, or inputs from a IBM PC/XT keyboard or mouse [35]. It contains the specific device drivers for these devices, as well as managing their interface to the OS-9 system. The GFM implements graphics primitives such as lines, circles, ellipses, area fill, and multiple text fonts of various sizes.

3.3.7.3 The RAVE Package Presentation Editor. The presentation editor (PE) is a tool for building user interfaces using the GSL and GFM [36]. Using a menu-based

interface, the application developer uses the presentation editor to select the appropriate interface objects and configure them for the intended use. The output from the PE is C source code, which can be linked and compiled with the application code and the necessary libraries to form the desired system.

3.3.7.4 The RAVE Package Interchange File Format Support Library. The interchange file format (IFF) support library provides the routines necessary to convert video and audio file into IFF format for transfer between dissimilar computer systems [37].

3.3.8 SRL-Provided Routines. The SRL-provided routines proved to be the most difficult to grasp. They were inadequately documented and spread over several directories. Essentially, SRL developed routines to do the following:

1. Configure the front end A/D converter and FIR filter
2. Find local peaks in the incoming signal
3. Transfer data between memories
4. Test the graphics display

Of these, only the front end configuration routine appeared to work correctly. The others either hung the system or caused a bus error and halted execution.

3.4 The System Memory Map

Understanding the addresses of the various boards in the system is key to properly programming the system. Complicating this understanding is the fact that multiple address spaces coexist in the PSK system. The system memory map in terms of global VMEbus

addresses is summarized in Table 1. Because of these multiple spaces, the layout of each board will be discussed first, then a summary of the address space for each board will be given as it relates to how the IV-3206, as system controller, addresses it. Because it is the system controller, the IV-3206 board will be discussed first.

Table 1. Global System Memory Map

System Memory Map		
Start Address	End Address	Device
0x0040 0000	0x0040 0012	I/Q Splitter Board Registers
0x0080 0000	0x00BF E000	Video Board RAM
0x0100 0000	0x0107 FFFE	a66540 FFT Board Registers
0x0200 0000	0x023F FFFF	Chrislin Dual-port RAM
0x0800 0000	0x083F FFFF	Local RAM for IV3206 board
0xFFFF E400	0xFFFF E406	IV-3272 Data Transporter Registers

The IV-3206 has its own address space, which the processor uses to do I/O, read and write memory, and access both the VMEbus and the VSB. The local 4 MB of RAM is addressed from 0x0000 0000 to 0x003F FFFF, with the lowest 10KB of addresses overlaid with interrupt handlers and reset vectors. This RAM can also be addressed by other VMEbus masters at VMEbus addresses 0x8000 0000 through 0xBFFF FFFF, and by VSB masters at VSB addresses 0x0000 0000 through 0x003F FFFF. The IV-3206 can address the VMEbus in A32 (extended addressing) mode from addresses 0x0040 0000 to 0xEFFF FFFF. The VSB is addressed on local addresses 0xF000 0000 through 0xFDFF FFEF. Local addresses 0xFE00 0000 through 0xFE3F FFFF are reserved for future use by Ironics. The Daughter Board Interface and EPROM (containing the monitor and boot loader) are addressed at addresses 0xFE40 0000 through 0xFE7F FFFF and 0xFE80 0000 through 0xFEBF FFFF, respectively. Local I/O ports are addressed at 1024 byte intervals in the range 0xFEC0 0000 through 0xFEFF FFFF. The VMEbus is addressed in A24 (standard

addressing) mode from 0xFF00 0000 through 0xFFFE FFFF, and in A16 (short I/O addressing) mode from 0xFFFF 0000 through 0xFFFF FFFF. This completes the IV-3206's 32-bit addressing space [20].

The addressing involved with the Chrislin 4 MB RAM board is much simpler. Since it is always a VME slave, it does not have a local addressing space with which to be concerned. It is addressed at both VMEbus and VSB addresses of 0x0200 0000 through 0x023F FFFF. Note that this falls into the global VMEbus address space used by the IV-3206; thus the external RAM board is directly addressable by the IV-3206, subject to VMEbus bus arbitration procedures.

The Vigra MMI250 graphics board memory address space is similar to, but not quite as complex as, the IV-3206 [23]. It resides at VMEbus base address 0x0080 0000 and extends to VMEbus address 0x00BF FA87, again within the global VME space addressed by the IV-3206. The 4 MB of video RAM occupies the bottom segment of that address space (0x0080 0000 to 0x00BF DFFF), with the rest of the space taken up with (in order from low to high addresses) the G300B graphics controller registers, the keyboard interface buffer and status register, the 68881 DUART control registers, the VME interrupt and mask registers, the VSB interrupt and mask registers, the G300B status register, and the SAA1099 sounds effects registers. The MMI250 can be addressed in either A32 or A24 modes with the same low order bytes; whether A32 or A24 addressing is used depends on the use of the address modifier codes of the VMEbus. The MMI250 RAM can also be addressed on the VSB at addresses 0xF080 0000 through 0xF0BF DFFF.

The Array Microsystems FFT board has a much simpler memory map. It resides at VMEbus address 0x0100 0000 through 0x0107 FFFE. Within that space, the board control and status registers occupy the bottom 64 bytes from 0x0100 0000 to 0x0100 003E, the programmable array controller registers occupy 0x0100 0280 to 0x0100 0298, and the processing memories occupy 0x0104 0000 through 0x0107 FFFE, all of which are also in the global VME addressing space of the IV-3206.

The IV-3272 Data Transporter board, unlike the other boards, is only addressable in the VME short I/O address (A16) space. It is jumper-configurable for one of four base addresses: 0xE000, 0xE400, 0xE800, or 0xEC00; in the PSK system, it is set to 0xE400. To the VMEbus, it appears as four registers: a control register at 0xE400, which configures, starts, stops, resets, and specifies the FIFO register data path width; the status register at 0xE402, which indicates the current status of the data transfer be processed by the IV-3272; the FIFO register at 0xE404, which is used to load transfer parameter blocks into the IV-3272; and the start/stop register at 0xE406, which actually starts and stops the transfer process. Because the IV-3272 resides in the A16 address space, the IV-3206 can address it at 0xFFFF E000 through 0xFFFF E006.

The SRL-constructed front end processing boards are also addressable on the VMEbus for the configuration purposes. The I/Q Splitter board is addressable as a series of registers starting at 0x0040 0000; these registers also configure the filters on the FIR filter board. As before, this series of addresses fall into the global VMEbus address space of the IV-3206. As noted previously, the front end processing boards use the VSB, but are not specifically addressable on it. Furthermore, they do not use the VSB in the manner prescribed by the VMEbus standard. However, they do not interfere with VSB interrupt

and bus arbitration because they are isolated from the general-purpose computing portion of the system using VSB jumper blocks located on the rear of the VME backplane.

3.5 The OS-9 Directory Structure

Like UNIX, OS-9 uses a tree directory structure and has specific default directories to contain various necessary software [29]. By convention, OS-9 directories are named using upper case letters to distinguish them from files. These default directories are outlined as follows:

- **C** – The C directory (in its only subdirectory, SOURCE) contains source code for user trap handlers and the startup routine linked into every C program by the linker when it is called by the C compiler executive.
- **CMDS** – The CMDS directory holds the executable files of the various OS-9 utilities. Its subdirectory BOOTOBS holds the relocatable object files containing device descriptors and drivers assembled from the source files in DEFS (see next entry).
- **DEFS** – The DEFS directory contains all the header files used to include libraries into C programs, both for OS-9 provided libraries and application libraries. This directory, as supplied by Microware, also contains device descriptor, device driver, and file manager assembly source files. These source files are made into relocatable object files and loaded into memory during bootup.
- **IO** – The IO directory contains the necessary file manager modules to interface with the graphics card. This directory has three levels of nested subdirectories isolating

drivers from descriptors first, then isolating the various types of modules (serial versus random block versus video).

- **LIB** – The LIB directory contains the various system libraries, including input/output libraries, trap handlers, and graphics support libraries.
- **MACROS** – The MACROS directory contains system-defined assembly language macro routines.
- **SYS** – The SYS directory contains miscellaneous system information files, including help files, system configuration source files, and the (unencrypted) password files used to validate user logins.
- **SYSSRC** – The SYSSRC directory (referred to as the SYSMODS directory in the documentation) contains the assembly language source files for the device descriptors and device drivers associated with serial interface devices and random block devices.

In addition, the directory structure created by SRL added the following directories:

- **A66540** – The A66540 directory contains four subdirectories: DEMO, DRIVERS, INCLUDE, and OBJECT. The DEMO subdirectory contains source files for an abortive development that tested the initialization of the array matrix processor on the a66540. Its makefile was improperly constructed and failed to link to the correct library. The DRIVERS subdirectory contains the source files for the board drivers discussed above as well as low-level assembly language drivers for board I/O. The INCLUDE subdirectory contains C language header files defining hardware-dependent constants and data structures. The OBJECT subdirectory contains the executable object modules produced by compilation of file in the other three subdirectories.

- **BEACH** – The **BEACH** directory contains C source files used by one of the SRL developers. Two of the source files are differing versions of an algorithm to find four local maxima over an input signal. Another source file implements block transfers in memory. Two other source files modify key assignments on serial terminals.
- **CLIPART** – The **CLIPART** directory contains two subdirectories: **CLUT8.LOW** and **CLUT8.HIGH**. Each of these subdirectories contains 8-bit color look up tables (CLUT's) used by the graphics file manager when creating displays on the graphics monitor. These CLUT's can be either low or high resolution, and are thus segregated in separate subdirectories for ease of use.
- **DEMO** – The **DEMO** directory consists of two subdirectories, **MYAPP** and **PERF**. The **MYAPP** subdirectory contains C source files generated by the tutorial contained in the **RAVE** package. The **PERF** subdirectory consists of similar files to those in **MYAPP**, but with extensions and other source files developed by SRL in their final user interface development effort. Some of these files may be adaptable to the current effort.
- **FRONTEND** – The **FRONTEND** directory contains the C source files used to load parameters into the front end A/D converter and FIR filter and the test code used to perform preliminary tests on the AFIT-constructed buffer memory. There are three slightly differing versions of the front end configuration and startup code, reflecting differing amounts of progress by SRL.
- **IV3272** – The **IV3272** directory contains source files to configure, operate, and test the **IV3272** data transporter. Two of the files (**dma.c** and **fft.c**) are identical, ex-

cept for the base addresses of the IV-3272 and IV-3206 boards used. Three others (dma.mail.box, dma32.c, and dma7_18.c) vary only slightly in the amount of in-line debugging code used. All three are used to test IV-3272 transfer characteristics.

- JUNK – The JUNK directory contains extra copies of system configuration modules.
- PCM – The PCM directory contains the necessary device descriptors and device drivers to run the floppy disk PC File Manager. This set of routines allows the connected floppy drive to read and write disks formatted in IBM compatible PC/MS-DOS format.
- POLY – The POLY directory contains more C source files generated by the RAVE presentation editor and two SRL-generated source files used to test and explore the RAVE system.
- PORT – The PORT directory has five subdirectories: BACK250, MMI100, MMI250, VIP640A, and VIRTUOSO. The BACK250 directory holds backup configuration files to allow a rebuild of the MMI250 device descriptors and drivers if necessary. The other four subdirectories contain the configuration files and startup files necessary to interface with their respective video boards (MMI100, MMI250, VIP640A, and Virtuoso). Since the PSK system only uses the MMI250 board, the other subdirectories could be deleted with no effect on system operation if disk space is needed (these files are available on the original distribution disks if needed).
- UTIL – The UTIL directory contains source code for extra OS-9 system utilities not documented in [29]. The files in this directory were successfully compiled and the resulting executable modules placed in the system CMDS directory.

- **UTILS** – The UTILS directory contains source code for utilities associated with the RAVE package. The files in this directory were also compiled and placed in the system directory. However, some of the utilities did not work correctly due to incorrect configuration of the makefile used for compilation. As a result, incorrect libraries were linked in, causing bus errors.

In addition to the directories added by SRL which remain, there are some directories and files which were removed due to redundancy or uselessness:

- **AM540** – The AM540 directory had been created apparently to hold variants of the A66540 drivers, but was unpopulated and therefore removed.
- **MOSSING** – The MOSSING directory, named after one of the developers at SRL, held only two C source files. One was a duplicate of a file contained in the BEACH directory, so it was deleted and the other file moved over to the BEACH directory.
- **FRONTEND** – The FRONTEND directory had a COPPOLA subdirectory used by the developer of the buffer memory. Many of the files in the directory were duplicates of files found in the parent directory; these files were deleted and the remainder moved into the parent, with the makefile dependencies adjusted accordingly.
- **SYSMODS** – The SYSMODS directory, although listed in [29] as a fundamental system directory, held only obsolete copies of source files otherwise stored in the SYSSRC directory. A check of the original distribution disks revealed no SYSMODS directory at all, so it and its redundant files were deleted.
- **Extra subdirectories of A66540** – These directories were created, but never used. To simplify the hard drive structure, they were deleted.

- Excess .login and .logout files – These files, used to set and un-set parameters during interactive logins and logouts, were found in multiple directories. Since only the .login and .logout found in the root directory are used, the excess files were deleted.
- Zero length files – Several files of length zero were found in the directory structure. Two of these files were used by makefiles to determine currency of compilation in their respective directories and were left in place; others, serving no apparent purpose, were deleted.

3.6 OS-9 System Configuration/Boot Sequence

Configuring the system is a highly flexible, but detailed, procedure. It involves creation of device descriptors, device drivers, and unique files used by OS-9 during bootup. Before discussing the configuration process, it will help to define the terminology used and briefly discuss the structure of OS-9; this will provide insight into both the why and how of the configuration process.

The OS-9 operating system is modular. The central part of the system, known as the kernel, does not know anything about the details of input/output devices. This modularity makes the system more robust and transportable across different hardware platforms. However, the system must be informed about the various I/O devices associated with the hardware on which it is running. This information is documented in a modular fashion. Device descriptors contain the information needed by the operating system in order to correctly interface with a particular I/O device, while still hiding the hardware-dependent details from the OS. The device driver, on the other hand, provides the low-level hardware dependent interface to a particular I/O device. The descriptor and driver are linked by use

of a file manager, which is common to classes of I/O devices. The classes of I/O devices used by OS-9 are Serial Communication Files (terminals, modems, mice), Random Block Files (hard and floppy disks), and Serial Block Files (tape drives, although the OMTI SCSI Controller tape interface driver appears to use the RBF file manager). The three types of modules taken together form the interface between the operating system and the I/O device. These modules are typically written in assembly code and assembled into object files before configuration can take place. In the PSK system, an additional file manager (the Graphics File Manager) is present.

Once the device descriptors, drivers, and file managers are assembled, the system configuration process can take place. The name given to the device descriptors during assembly (typically done using a makefile) become the name by which that particular device is known to the system. In other words, the hard disk device descriptor is assembled into a module named h0; after bootup, the hard disk is accessed as /h0. The modules which must be loaded into the system are linked together into a single file with the name OS9Boot. Furthermore, the first module in that file must be the init module. It is assembled from the assembly language file init.a, and warrants special mention.

The init.a file sets the essential system constants and definitions. It is in this file where the system memory map is laid out, either directly or by inclusion of the file oskdefs.d. The PSK system has several different init.a files on it, reflecting different stages of development and different possible overall system configurations. The file currently in use is the file located in directory SYSSRC.

The OS9Boot file is not the only file which controls system configuration. It is analogous to the CONFIG.SYS file found on IBM compatible microcomputers. There is another file, **startup**, on OS-9 systems which is analogous to the AUTOEXEC.BAT file on IBM compatible microcomputers. The **startup** file contains commands which are executed during the boot process and can be used to load additional modules into memory, copy files, or execute other OS-9 commands.

The OS-9 boot process, then, is as follows: the system loads the modules contained in the file OS9Boot, executes the commands found in file **startup** (if any), then awaits a command on the console terminal. If the timesharing monitor utility **tsmon** has been executed by **startup**, then an interactive login will be required, with password. Once successfully logged in, the user begin execution of any legal OS-9 command or application.

In the PSK system, the boot process initializes the system processes and device drivers for the serial ports, video card, keyboard driver, mouse driver, and disk drives. One important initialization is that of a RAM disk. The RAM disk is initialized to mirror the critical portions of the hard drive: parts of the DEFS, LIB, RELS, and SYS directories on the hard disk. The system libraries, system information files, and include files for the C compiler are copied to this RAM disk and used during program development, if the necessary makefile includes the proper paths. Using the RAM disk for program development does speed up compilation, but results in errors or inclusion of the wrong libraries if improperly used.

3.7 Requirements Analysis

Now that the PSK system's capabilities are generally understood, the next step is to define the essential functions that the system needed to perform. These functions are later used in Chapter IV for the software analysis and design phases to define what software needs to be developed. During a typical software development process, users are extensively surveyed and consulted to determine system requirements and in fact are the primary driving force behind system development. For the PSK system, some of the overall system requirements were specified by the task order given to SRL [1], which in turn determined the system configuration built by SRL. For the purposes of this development, then, it was necessary to substitute domain-knowledgeable thesis committee members and other interested parties for the real-world users who would ordinarily be a part of the process. Discussions with these "users" revealed the following requirements for the system:

3.7.1 User Interface. The user interface shall be a graphical interface. Users will have the option of selecting commands from a menu system or typing at a command line. The menu will be accessible through keyboard commands or by using a mouse. The user will not have the option of customizing the overall basic format of the interface.

3.7.2 Signal Processing Commands. The user will have the following signal processing services available:

- display the contents of a file, either as raw numbers or a graph.
- take a "snapshot" of incoming real-time data and write it to a file or display it as a graph.

- perform a FFT, with the size (number of points) interactively specified, with the input selectable from the real-time input stream or a file.
- direct the output of the FFT to the screen, a file, or both, at the user's option.
- select the FFT board which performs the FFT (either the a66540, the TMS320C25 on the IV3272, or another user-installed board).
- select the operating parameters/filter coefficients for the front-end FIR filters.

3.7.3 System Commands. The user will have the following system services available, with the specific syntax of OS-9 hidden:

- displaying the contents of a directory
- moving around in the file/directory system
- displaying current working directory (data and execution)
- copying files
- listing (printing) files
- creating and deleting directories
- creating, editing, and deleting files
- list/modify file attributes
- finding the directory location of a particular file
- display system date and time
- display free space on the disk drives or system memory
- obtain system help

- transfer a text file to the PSK system from another system, or the reverse

3.7.4 Operational Concept. A "normal" user will use the VT-340 terminal to login to the system. Once logged in, the user will execute the signal processing software, and will only use the auxiliary keyboard, mouse, and NEC 5D graphics display for interaction with the interface described above. Superusers (i.e., system developers) will have access to all system functions, including the full range of development tools and OS-9 commands. "Normal" users will not have access to the OS-9 system commands, the C compiler, and the user interface development tools.

3.8 Summary

This chapter detailed the current hardware and software configuration of the PSK system, outlined key OS-9 and memory concepts critical to successful software development on the system, and established the requirements the overall software must meet to produce a workable system. The chapter first discussed in detail the hardware components which make up the system, including the real-time signal processing components and the general purpose computer modules. It then discussed the structure and functionality of the original software provided by the hardware manufacturers and the limited amount of software provided by SRL. It also discussed the memory structure of the system and outlined the different methods required to access the different hardware modules via the VMEbus and the VSB. Configuration and initialization of the OS-9 operating system were also discussed, focusing on the concepts necessary for software development and understanding. Finally, a detailed list of requirements along with the process used to develop the list were discussed.

IV. Software Analysis and Design

4.1 Introduction

This chapter outlines the process and details associated with the development of object-oriented analysis and design models for the user interface system. The first part of the chapter describes the analysis models, which are of three separate types: the object model, which models the relationships between the component pieces of the system; the dynamic model, which illustrates the behavior of the parts; and the functional model, which focuses on the data interchanges between the objects. The second part of the chapter describes the design models. Like the analysis models, the design models consist of the object, dynamic, and functional models and detail the same categories of information. The design models, however, are at a deeper level of detail to facilitate direct translation into a source code implementation. The chapter ends with a summary of the development process and its results.

4.2 Analysis Process and Models

According to Rumbaugh [12], the analysis process is used to clarify and refine the requirements stated by the user in the requirements documents. This clarification often requires multiple iterations of the analysis models and documentation. Analysis of the software requirements for the PSK user interface system required just such an iterative process; the Rumbaugh object-oriented modeling technique (as documented in [12] and discussed in portions of the AFIT software engineering curriculum [38]) was used for the entire analysis and design process in this thesis.

The analysis models shown later in this chapter are the result of that process. Since this chapter was not intended to be a tutorial on the object-oriented development process, not all of the iterations of the models will be shown. The first object model developed will, however, be shown since it is useful for understanding the relationships between the hardware and software entities included in the system. This initial model is shown in Figures 3, 4, and 5. In this model, each hardware module is shown as a separate object, with subsystems identified by the use of the aggregation symbol. The attributes of each hardware object (i.e., the parameters which must be loaded into it or used by software for proper operation) and the operations performed by the hardware are shown as the object attributes and operations. Each major software subsystem is also shown with appropriate attributes (parameters) and available operations.

This initial model, while complete and highly detailed, proved unworkable for software development. While it did provide some abstraction, it included too many implementation constructs to be useful for the analysis process. It did, however, provide a starting point for development of a better analysis model. The top level of objects were "stripped off" from the initial model and used as a baseline for development of the object model. Furthermore, this initial model highlighted a unique aspect of this development: many of the objects to be modeled (in fact, all of the hardware objects) have only one instantiation - for example, there is only one instance of the object class "Real-Time Front End Processor" in the system. The effects of this property on design and implementation are discussed later.

Next, the finalized analysis models are discussed. The discussion is presented from the point of view of developing an initial set of analysis models; however, the models

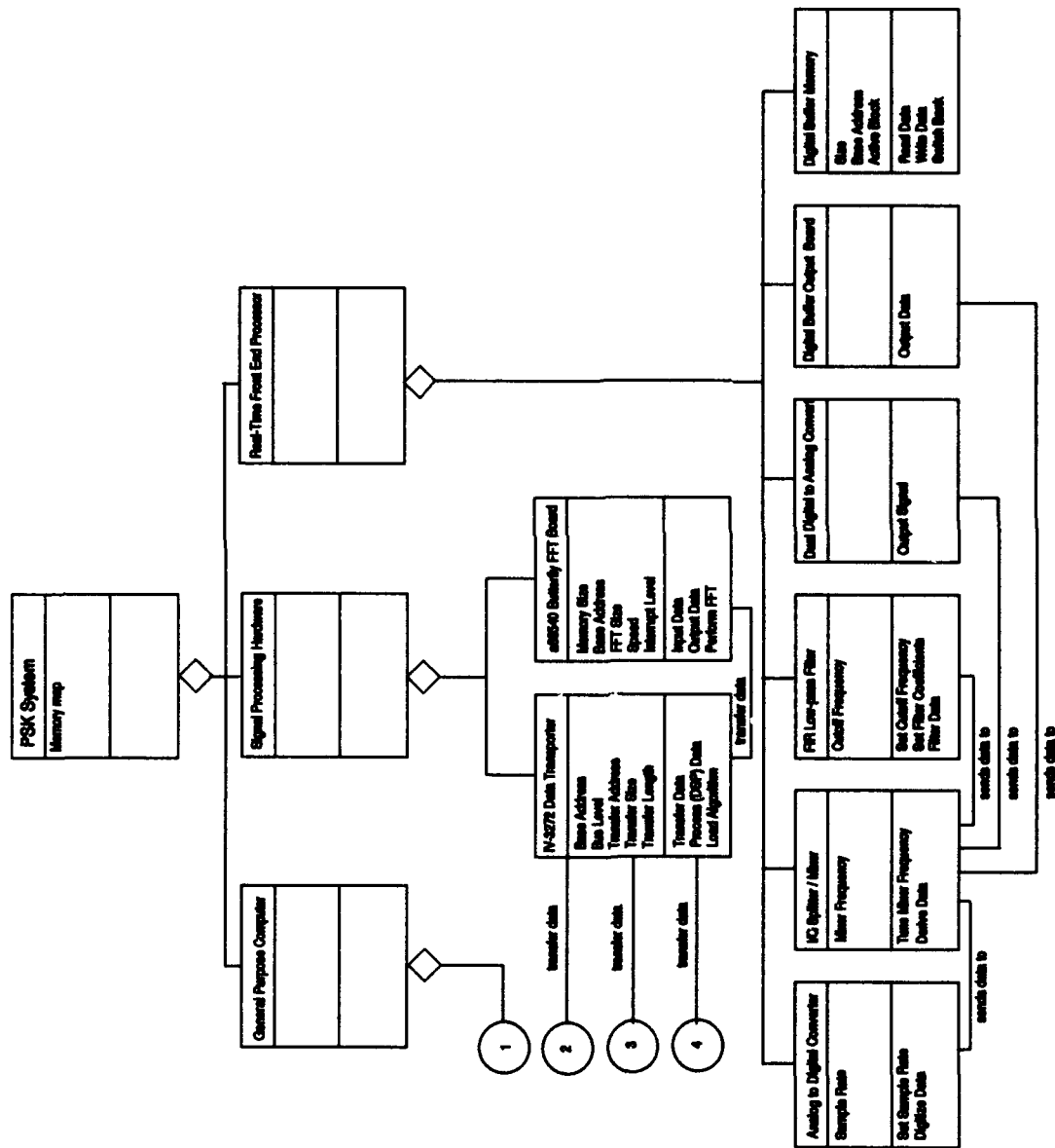


Figure 3. System Object Model - Major Subsystems and Front End / DSP Hardware



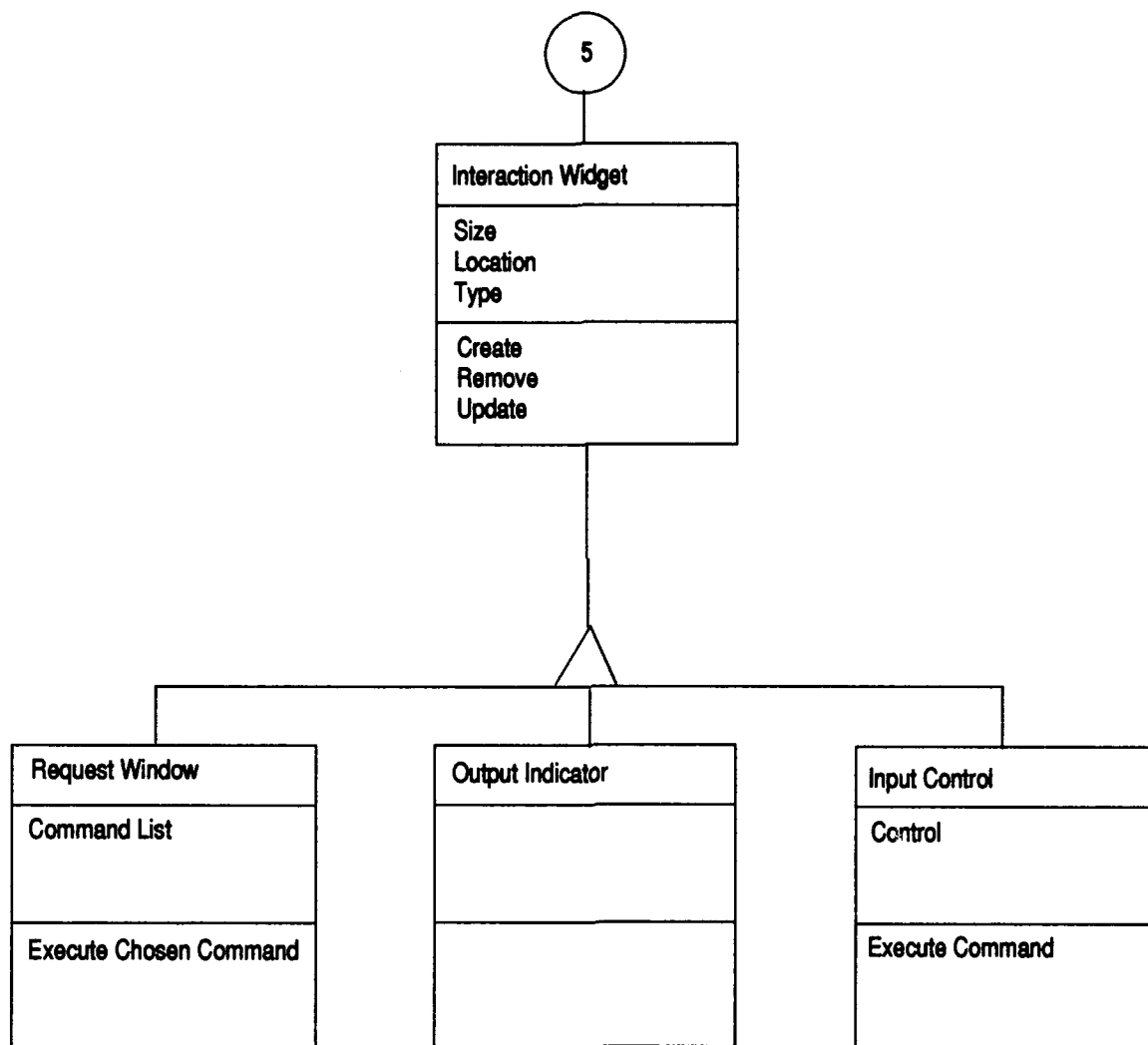


Figure 5. System Object Model – User Interface Software Objects

discussed are actually the final models developed. The first model discussed is the Object Model and is shown in Figure 6, using the OO notation of Rumbaugh [12]. In the diagram, the multiplicity of an association is shown by the presence or absence of a dot at each end of the line denoting that association. If a dot is present, the association is of multiplicity one or more; otherwise, an association of exactly one is indicated by the absence of the dot. The "consists of" relation is shown by a diamond, and the superclass/subclass relation is denoted by a triangle linking the subclasses to the superclass.

4.2.1 Object Model. Because it represents real-world entities, the object model is the starting point for analysis. The object model forms the basis for the dynamic and functional models by capturing the relationships between parts of the system; these relationships will identify events that affect multiple objects and data flows between the objects, two key analysis tools in the development of the dynamic and functional models.

4.2.1.1 Object Class Identification. The first step in building the analysis object model is to identify the relevant real-world object classes. This was accomplished by reviewing the specification and building of list of the nouns contained in it, then removing duplicate classes, classes which appeared to be attributes or operations, and pure implementation constructs. The system object model previously built was also used, as discussed earlier in the chapter. For the PSK system, the remaining object classes turned out to be the following, listed in alphabetical order:



Figure 6. Software Analysis Object Model

- **Analog Input Signal** – This object class models all analog input signals to the Real-Time Front End Processor class. There are potentially an infinite number of instances of the object class, but only one instance is permissible to be present at a given time.
- **Discrete Digital Signal** – This abstract object class models all digital signals present in the system, and has two concrete subclasses corresponding to the domain the signal exists in, either the time or frequency domain. These signals are also discrete in amplitude to match the quantized signal produced by the Real-Time Front End or the input signals generated by the user.
- **Display** – The Display object class models the graphical display. There is also only one instance of this object class, representing the NEC monitor.
- **Frequency Domain Signal** – Frequency-domain signals are those quantized signals which have been transformed from the time domain by virtue of the Discrete Fourier Transform or produced by filtering of another frequency-domain signal. In the PSK system, both of these actions are accomplished by a member of the Signal Processing Hardware object class. Multiple instances of this object class can exist simultaneously; however, there is a one-to-one association between a particular instance of this class and a particular instance of the Time Domain Signal object class.
- **Input Device** – The Input Device object class models the keyboard and mouse. There will be two instances of this class for analysis purposes.
- **Real-Time Front End Processor** – The Real-Time Front End Processor object class models the five SRL-constructed boards. The details of those individual boards were combined into this object class and are treated as a single system accepting analog

input and producing two digital output streams (the I/Q bit streams). There is also only one instance of this object class in the system.

- **Signal Processing Hardware** – The Signal Processing Hardware object class models all hardware items capable of performing DSP operations: the a66540 board, the IV-3272 board, and any user-added boards installed at a later date. There will be multiple instances of this object class coexisting in the system.
- **Signal Sample** – The Signal Sample object class represents the individual samples of a Discrete Digital Signal.
- **Time Domain Signal** – Time-domain signals are those quantized signals which have been generated by sampling an analog input signal or by filtering of another time-domain signal. Again, both of these operations are performed by a member of the Signal Processing Hardware object class. Multiple instances of this object class can also exist in the system simultaneously, subject to the restriction listed above for objects in the Frequency Domain Signal class.
- **User Interface** – The User Interface object class models the overall control of the system. There is only one instance of this object class. To avoid premature design or implementation decisions, the format and interaction style of the User Interface is not specified at this time.

4.2.1.2 Association Identification. Associations identify the key relationships between objects. These relationships can be invariant relationships which must always hold true, show communications between objects, or both [12]. On the object diagram, the associations are shown as lines connecting the object classes and an association

name chosen to convey the nature of the relationship. These associations and the objects they connect are discussed next.

- **Controls** – The “controls” association relates the User Interface with the Real-Time Front End and the Signal Processing Hardware object classes. It represents the flow of commands and information from the user interface to the controlled objects and the return of status information from those objects.
- **Fourier Transform** – The “Fourier transform” association represents the mapping between a Time-Domain Signal object and a Frequency-Domain Signal. It is the result of the Discrete Fourier Transform operation performed to transform a finite time-domain signal into a finite frequency-domain signal.
- **Is Displayed On** – The “is displayed on” association relates the User Interface, Analog Input Signal, and Discrete Digital Signal object classes with the Display. It shows the necessary flow of information to the graphical display for user output and captures those objects intended for actual display, as opposed to internal information used by the system not intended for display.
- **Processes** – The “processes” association captures the dual nature of the relation between the Signal Processing Hardware and the Discrete Digital Signal object classes. The Signal Processing Hardware can take an object of either Discrete Digital Signal subclass as input and produce an object of either the same or the other subclass of Discrete Digital Signal as output, subject to the parameters supplied to the Signal Processing Hardware. This association could be modeled in greater detail, but was left simplified for the analysis portion of the development process.

- **Produces** – The “produces” association represents the relation between the Real-Time Front End Processor and the Time-Domain Signal subclass of Discrete Digital Signals. Every instance of an Analog Input Signal can have one or more corresponding instances of a Time-Domain Signal. These differing instances arise as a result of differing parameters loaded into the Real-Time Front End Processor. For example, the same Analog Input Signal can produce several Time-Domain Signals, each corresponding to sampling at a different sampling rate. This relationship is captured by the combination of the “Produces” association and the “Samples” association, which is discussed next.
- **Samples** – The “samples” association relates the Analog Input Signal to the Real-Time Front End Processor and captures the nature of the function performed on the Analog Input Signal.
- **Takes Input From** – The “takes input from” association connects the User Interface with the Input Device; it shows the communication of information from the user to the user interface.

4.2.1.3 Attribute Identification. Attributes identify the important characteristics of an object class. They can be used to identify instances of an object class; however, it is possible for two object instances to have identical attribute values and still be unique. For each object class, attributes were identified which would serve to define and determine the state of the object. For each of the objects identified in the object model with identified attributes, the attributes are listed and defined:

- **Analog Input Signal** – The Analog Input Signal has three attributes: **Bandwidth**, which is the difference between the highest and lowest frequencies found in the signal; **Length**, which is the time difference between the beginning and ending of the signal; and **Maximum Amplitude**, which is the highest value found in the signal.
- **Discrete Digital Signal** – The Discrete Digital Signal has three attributes: **Number of Samples**, which is the number of discrete time or frequency intervals in a particular sample; **Maximum Amplitude**, which is the highest sample value found in the sample; and **Domain**, which denotes the domain (i.e., time or frequency) of the signal.
- **Display** – The Display has three attributes: **Length**, the vertical measurement in pixels; **Width**, the horizontal measurement in pixels; and **Resolution**, the distance between pixels.
- **Real-Time Front End Processor** – The Real-Time Front End Processor has five attributes: **Base Address**, the VMEbus address at which the configuration registers are found; **Sample Rate**, the inverse of the time interval between samples; **Mixer Frequency**, the frequency of the mixer on the I/Q splitter board; **Cutoff Frequency**, the low-pass cutoff frequency of the FIR filter board (a derived attribute); and **Filter Coefficients**, the set of numbers loaded into the FIR filter board to produce the desired Cutoff Frequency.
- **Signal Sample** – The Signal Sample has two attributes: **Sample number**, which is a serially-assigned integer representing the location of the Sample in the sequence of Signal Samples; and **Sample Value**, which is the magnitude and phase of the Signal Sample.

- **Signal Processing Hardware** – The Signal Processing Hardware has five attributes: **Memory Size**, the size in bytes of the on-board memory; **Base Address**, the VMEbus address at which the configuration registers are found; **FFT Size**, which is the length of the FFT performed by the hardware; **Speed**, which is the time required to perform the desired DSP function; and **Interrupt Level**, the VMEbus interrupt used to signal completion of a DSP task.

The complete object model includes the descriptions of the attributes and object classes given above (the data dictionary for the model) in addition to the diagram shown. This model must be expanded to include design artifacts during the design phase. Now, however, the desired behavior of the system will be explored and documented in the dynamic model.

4.2.2 Dynamic Model. The dynamic model focuses on the event-driven behavior of the system. For the PSK system, this is a critical model, since it is intended to be used in a highly interactive manner and eventually be used for analysis of real-time signals. The system must be capable of processing these signals in the time available before the next portion of the signal of interest arrives. In essence, this requirement means that the system must continuously process each portion of these signals at such a rate that buffers are not overwritten by new incoming signals. Fortunately, in this system, the hardware provided operates at a high enough speed that it ensures timely processing will take place due to the dedicated signal paths provided[25]. Other than capturing incoming signals, the requirements specification developed in Chapter III does not specify any real-time signal processing constraints. All specified processing is performed on previously captured files,

and the only signals with time constraints are those signals which are being captured and either displayed, written to disk, or both.

Just as important to requirements satisfaction as correct timing is correct event response of the system, which is also modeled by the dynamic model. The system must respond appropriately to external events in order to meet the specified requirements. These responses can be summarized in scenarios, which are discussed next.

4.2.2.1 Scenario Preparation. Scenarios are typical interaction sequences between the user and the system. They can be thought of as a series of events [12]. Scenarios covering normal (non-error condition) interaction sequences are prepared first, then they are analyzed to determine where errors can be introduced. The scenarios are prepared by examining the user requirements specification, then listing the events which must occur to satisfy those requirements. In the scenarios developed, each normal scenario trace is grouped with its corresponding error-handling actions. For each scenario, the object or other stimulus (the user, via the input device, in most cases) which causes the event is listed along with the parameters associated with the event. The viewpoint is that of the user, who sees the system as a single entity. The system is modeled at this point as a generic computer with the functionality provided by the hardware of the PSK system. A sample scenario so developed is shown in Table 2; the complete list is contained in Appendix A.

4.2.2.2 Event Identification. The scenarios discussed previously can be used to identify system events which must be communicated between objects in order to successfully complete the scenarios. Each scenario is traversed to find the events, then the

Table 2. Example Scenario

Scenario 1 – User wants to execute an operating system command with parameters
User selects “Execute operating system command” from list of available commands
System responds with list of choices
User selects desired action
System asks for necessary parameters based on the action desired
User supplies parameters
System executes command and returns to main list

events are grouped by the objects affected. The following events were identified from the scenarios listed in Appendix A:

- Display
 - display user interface menu
 - display signal
 - display samples
- Real-Time Front End Processor
 - configure processor
 - start processing/sampling
 - stop processing/sampling
- Signal Processing Hardware
 - configure hardware
 - read data
 - write data
 - perform Fast Fourier Transform
- User Interface
 - receive input from user
 - validate user input
 - cause menu to be displayed
 - send data to Real-Time Front End Processor
 - send data to Signal Processing Hardware

4.2.2.3 Event Flow Diagram Preparation. The event flow diagram is derived from the listed events and scenarios. It is intended to aid in the development of the final state diagrams which model system behavior by showing the interactions between the various objects citerumbaugh. Events which involve only a single object are not shown. The event flow diagram for the system is shown in Figure 7.

4.2.2.4 State Diagram Development. The state diagrams are the final product in the dynamic model. They summarize and document the time-dependent behavior of the system and its reactions to external stimuli. Each state is denoted with a descriptive state name and a series of actions to be performed while in that state. State transitions occur either as a result of an event, or automatically at the completion of all state activities. Some transitions are guarded (shown as [*guard*]) so that the transition will not occur unless both the requisite event has occurred and the guard condition is also satisfied. Not all objects have a state diagram. Some objects do not have interesting state behavior; for example, they are initialized, perform an action, and then return to an idle state or are terminated. On the other hand, an object need not be actively operating on other objects in order to have interesting state behavior. The primary action objects in the PSK system are the User Interface, the Signal Processing Hardware, and the Real Time Front End Processor. These objects have state diagrams shown in Figures 8, 9, and 10, respectively.

The state diagram for the User Interface Object shows that the User Interface Object is essentially acting as a dispatcher, selecting which system function will be performed based on which top level choice the user makes. It has an explicit initialization and termination state to ensure a clean system startup and shutdown. The Signal Processing Hardware

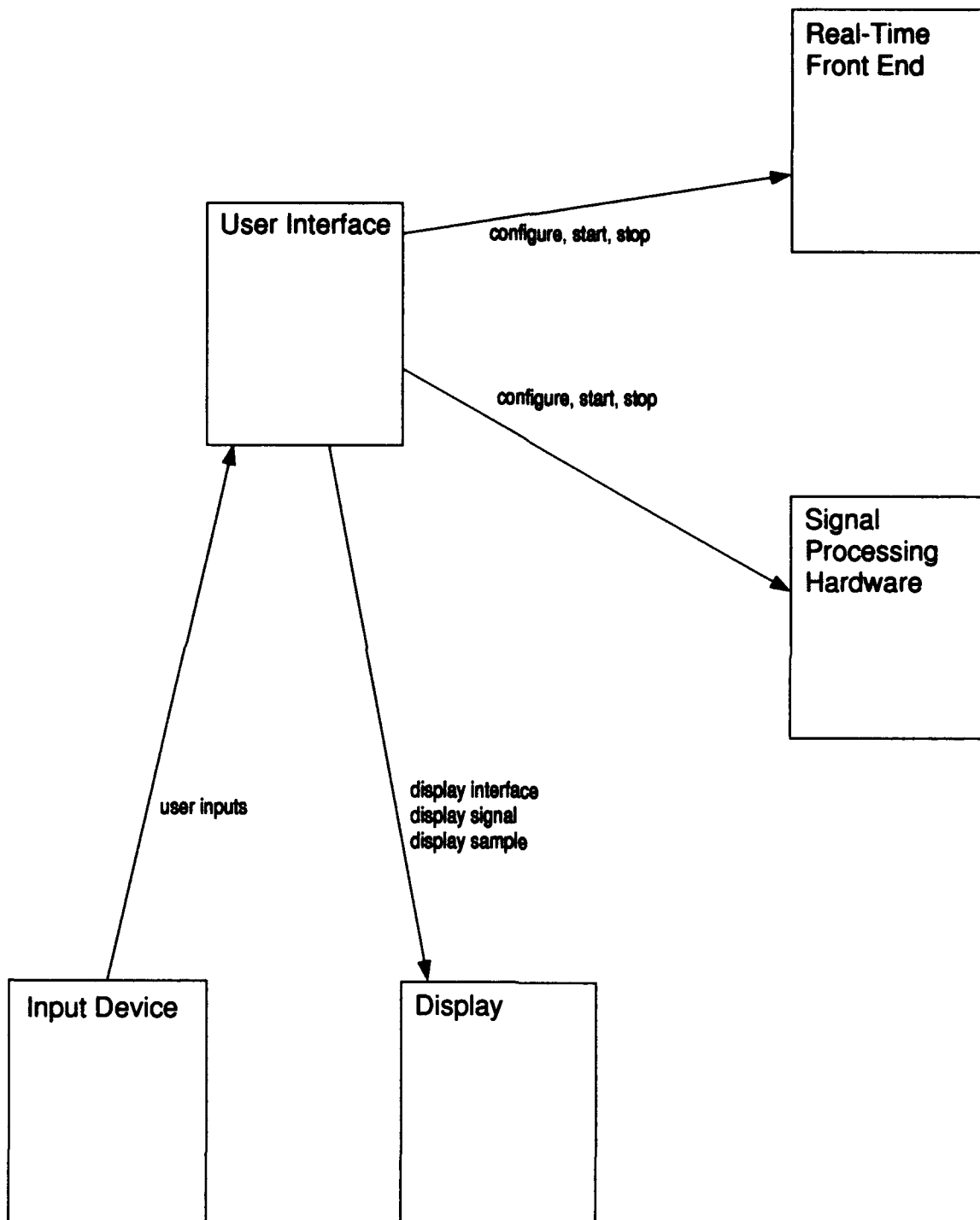


Figure 7. Event Flow Model

Object basically moves between two states. The Hardware is either idle (and configured) or busy processing a signal. The Initialization state is provided to catch attempts to start processing a signal without being initialized first. The Real-Time Front End Hardware Object has state behavior similar to that of the Signal Processing Hardware. This is expected: this object must also be configured before it can start processing/sampling an input signal; once configured, it is either waiting to begin processing or is actually processing a signal. Every event which is sent by an object is received by another object. This communication is shown by the matching event names on the state diagrams.

4.2.3 Functional Model. The functional model describes the data transformations which the system must perform. It specifies what the system must do, but not necessarily how it is done. The events which occur to make it happen are specified on an indirect basis. It is cast in terms of inputs, outputs, and the transformations which must occur to turn the input into the output. It makes sense, then, to first identify the relevant inputs and outputs to the system to begin development of this model. The functional model is documented as a combination of the data flow diagram and the descriptions of the process bubbles contained in the data flow diagram.

4.2.3.1 Input and Output Identification. Inputs to the system are those data items needed to successfully perform the desired functions. Control inputs, those inputs which solely determine what is to be done, are not listed here; they are taken into account in the dynamic model. The inputs to the PSK system are, in alphabetical order, as follows:

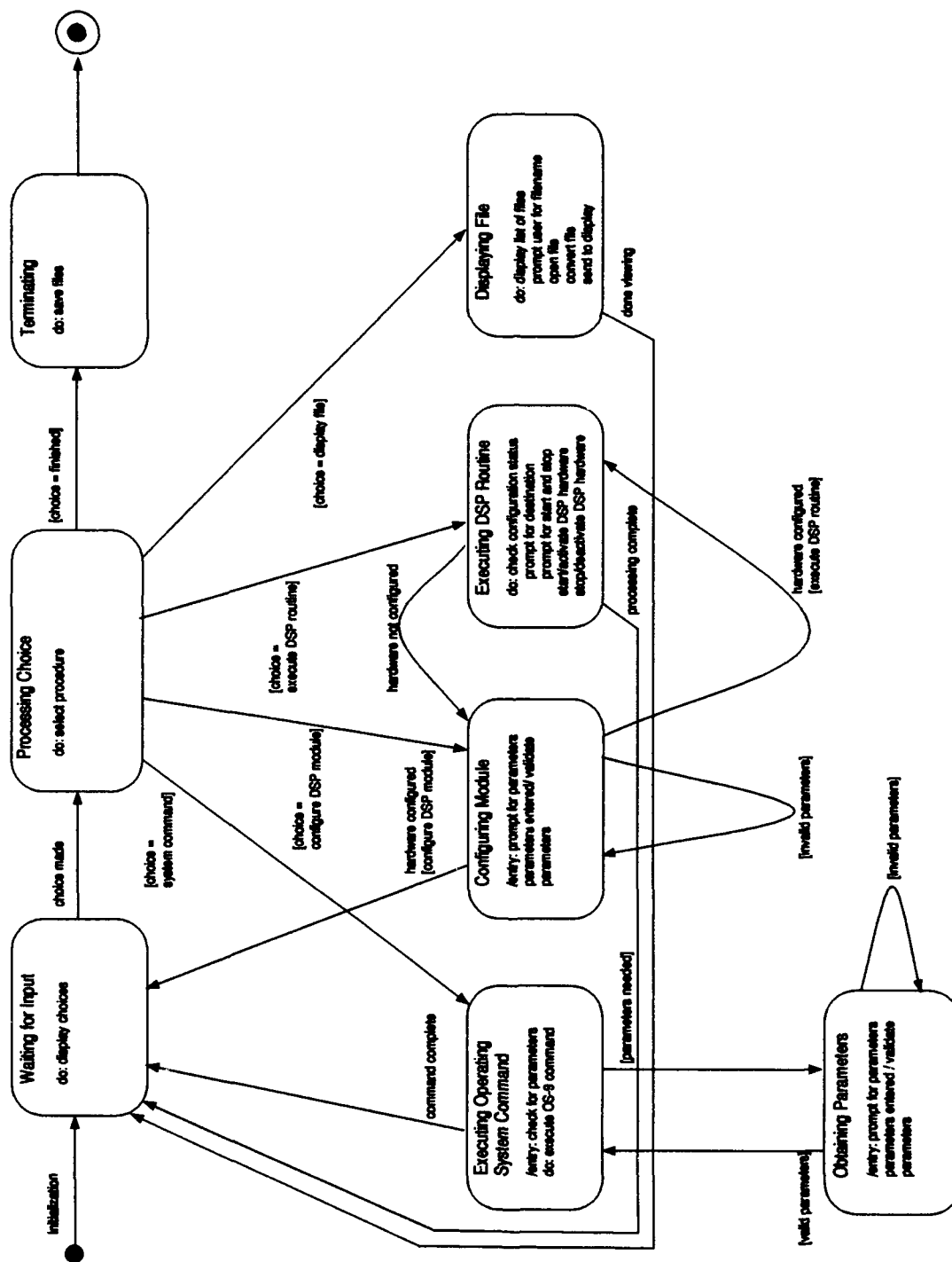


Figure 8. Software Analysis Dynamic Model – User Interface State Diagram

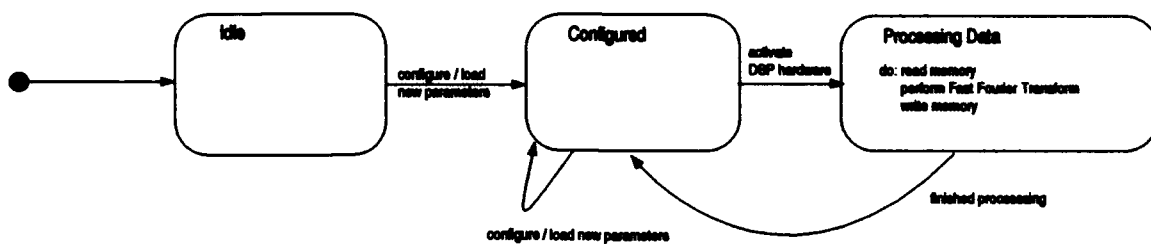


Figure 9. Software Analysis Dynamic Model – Signal Processing Hardware State Diagram

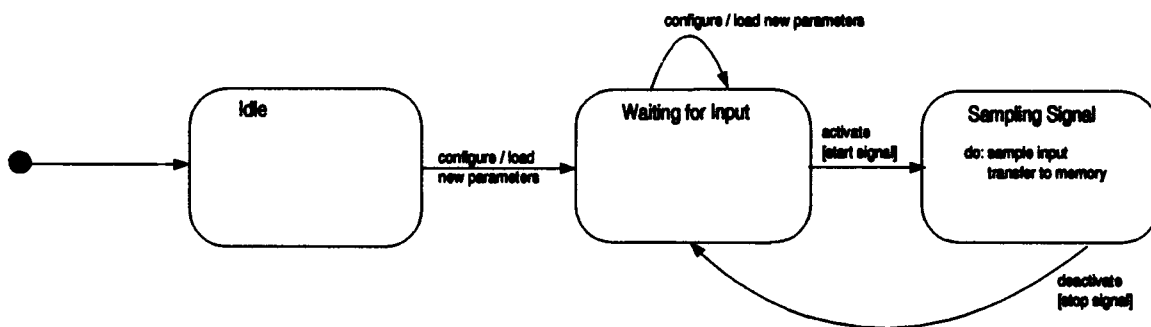


Figure 10. Software Analysis Dynamic Model – Real Time Front End Processor State Diagram

- **Analog Input Signal** – This is the actual signal for which analysis is desired. It is supplied to the Real-Time Front End Processor.
- **Discrete Digital Signal File** – This is the formatted file used to store the Discrete Digital Signal.
- **File Name** – This is the name of the file used for either input or output, as known to the system.
- **Filter Parameters** – These are the coefficients used by the front end FIR filter to set the cutoff frequency of the filter.
- **Plot Size** – This is the number of sample points used for the signal plot.
- **Sample Size** – This is the number of sample points used for the FFT process.
- **Signal Values** – These are the actual amplitude and phase values representing the sampled signal (either time- or frequency-domain) at each sample interval.
- **System Command** – This is a natural language command indicating a desired operating system function (for example, “Show contents of current directory”).
- **System Command Parameters** – These are any parameters required to complete execution of a System Command.

The outputs from the PSK system are as follows:

- **Discrete Digital Signal File** – This is the same item as listed under “Inputs”.
- **Sample Listing** – This is the ordered list of samples listed in sample number ascending order.

- **Signal Graph** – This is a rectangular-coordinate system representation of the sampled signal in either the time or frequency domains. The sample number is listed along the abscissa and the magnitude of the signal is along the ordinate.

4.2.3.2 Data Flow Diagram Development. Once the inputs and outputs were identified, diagrams were developed to show what processing and transformations needed to happen to change the inputs into the outputs. For the PSK system, there is one primary data flow diagram, which details the transformations on the inputs. There is another minor data flow diagram showing the input to the operating system interface, but the detailed output from this transformation is not shown since it is used internally by the system. The data flow diagram is shown in Figure 11.

4.2.3.3 Functionality Description. The transformation “bubbles” in the data flow diagram are named so that their functionality is concisely described. However, to validate the analysis, their functionality must be described in greater detail to ensure the proper transformation is indeed taking place. The transformations are described in the following paragraphs, listed in alphabetical order:

- **Convert Signal to Plot** – This transformation changes the time or frequency domain signal output into the (x,y) pixel information needed by the display object and adds scaled axes for reference. It reads the number of digitized signal points specified by Sample Size and plots on the Display the magnitude and phase of each sample point. It also plots scaled axes for reference.

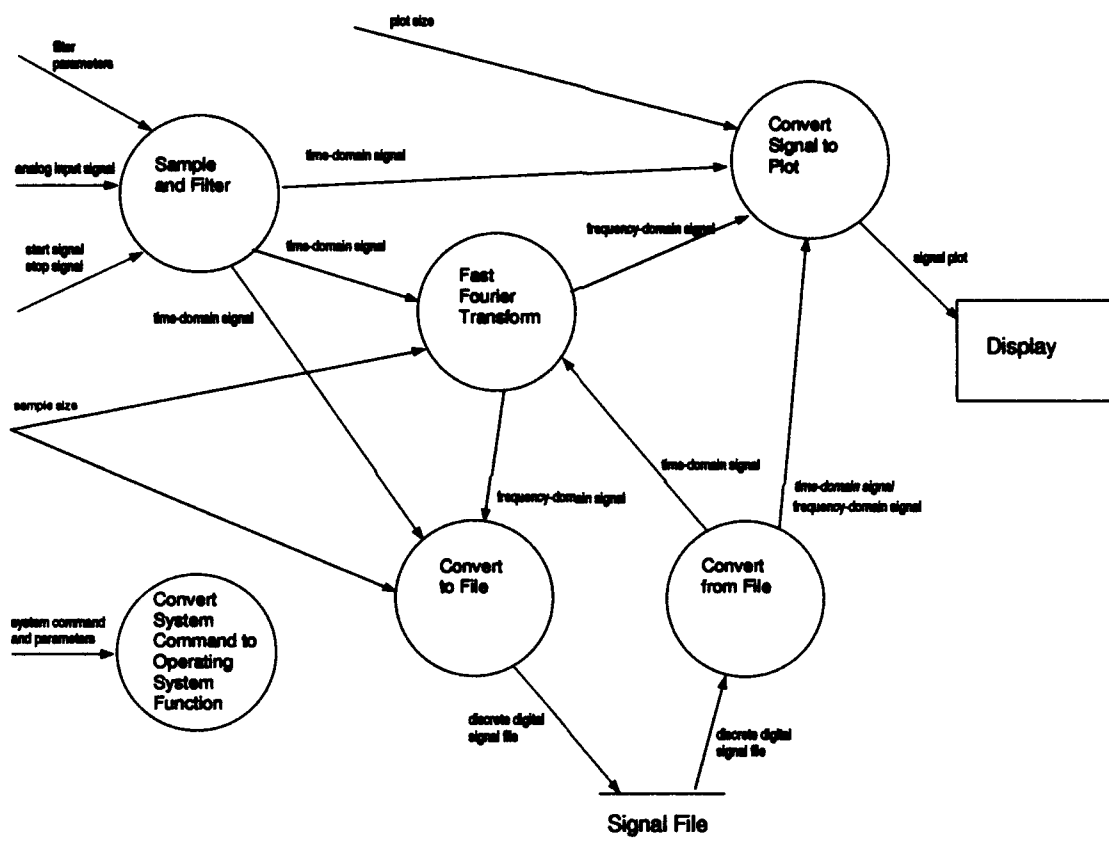


Figure 11. Software Analysis Functional Model – Data Flow Diagram

- **Convert System Command to Operating System Function** – This transformation interprets the natural-language system command given by the user into the operating system function or functions required to execute that command, using the parameters supplied by the user.
- **Convert from File** – This transformation reads from a file the number of signal points specified by (Sample Size) in the discrete digital signal and transforms them into the form required for manipulation by the Signal Processing Hardware.
- **Convert to File** – This transformation formats the number of signal points specified by (Sample Size) in the discrete digital signal into the form required for storage in a system disk file.
- **Fast Fourier Transform** – This transformation takes as an argument the Sample Size and performs the Discrete Fourier Transform on a time-domain digital signal, which converts it to the equivalent frequency-domain digital signal.
- **Sample and Filter** – This transformation executes the following steps. When commanded to start sampling, it samples the information contained in the analog input signal until the desired number of points have been obtained or a stop sampling command is received. The bandwidth of the analog input signal sampled is determined by the hardware, and is based on the filter parameters provided prior to the start of sampling (during configuration).

At this point, the analysis is complete. The next step in the development process is the creation of the design models. This stage of development is described in the next section.

4.3 Design Process and Models

The design process is the method by which the highly-abstracted models developed during the analysis phase are turned into realizable models easily translated into a programming language. As in the analysis model, a strategy must be employed to fully develop the final design model. The design process used here is a combination of the design strategy used by Rumbaugh [12] and the strategy described in portions of the AFIT software engineering curriculum [38]. As before, this process is an iterative one, requiring several passes to converge on the best solution. The design process includes the following steps:

- Combine the three analysis models to determine operations on objects – This step lays the groundwork for design by associating the objects developed in the analysis phase with operations. The operations are identified by examination of the dynamic and functional models. The dynamic model identifies which objects are involved in an operation, and the functional model identifies the essential nature of the operation. For operations that affect multiple objects, the object with which the operation is associated is a design decision.
- Define the User Interaction style – Until this point, the behavior of the user interface has been treated at a high abstraction level. Little mention has been made of specific interaction modes and techniques. This step will define those modes and techniques using the guidelines defined in [14] and implement behaviors as specified by the analysis models.

- Add lower level and interface objects – This step adds detail to the models developed during analysis. Device-level interfaces and procedures which were abstracted away during analysis are now incorporated into the design model.
- Define algorithms and data structures needed to implement the operations – This step is where the operations are transformed into the pseudo-instructions and data models which will eventually be coded in a programming language during implementation.
- Define how control will be implemented – This step determines how the flow of control will be handled: either as a procedural method, a series of communicating state machines, or a set of concurrently running tasks.
- Update the analysis models – In this step, the analysis models are updated to include the detail developed during the design stage. These models, along with the algorithmic and data structure information developed, form the documentation basis for implementation.

4.3.1 Analysis Objects Operations Addition. During this portion of design, the dynamic and functional models were compared to identify object operations. The event flow diagram was also used to assist in identification of necessary operations and primarily served to separate the initiator from the target of the operation for those instances where an action spawned operations in multiple objects. At this point, the identified operations are still not fully defined, and may require further operations on lower-level objects yet to be identified. The operations identified at this level, and their descriptions, are as follows:

- **Analog Input Signal**

- **Sample** – This is the actual process performed by hardware which samples the signal

- **Discrete Digital Signal**

- **Display** – This operation reformats the Discrete Digital Signal for presentation on the User Interface. It will be inherited by the two domain-separated subclasses.

- **Display**

- **Present Information** – This is the actual operation performed by the hardware which displays the User Interface on the screen.

- **Frequency-Domain Signal**

- **Inverse FFT** – This operation transforms the Frequency-Domain Signal into a Time-Domain Signal.

- **Input Device** – No operations of interest were identified that are controllable within the interface

- **Real-Time Front End Processor**

- **Set Sample Rate** – This operation obtains the desired sample rate and loads it into the analog-to-digital converter
 - **Set Mixer Frequency** – This operation obtains the desired mixer frequency and loads it into the I/Q splitter and mixer

- Load Filter Coefficients – This operation obtains the desired low-pass cutoff frequency, calculates the necessary FIR filter coefficients, and loads them into the FIR filter
 - Start Sampling – This operation begins the sampling process by turning on the front end hardware
 - Stop Sampling – This operation ends the sampling process by turning off the front end hardware
 - Validate Parameters – This operation examines the parameters and determines if they will cause acceptable operation of the Hardware
- Signal Processing Hardware
 - Set Configuration – This operation obtains configuration data from the user, then creates the configuration data blocks required by the Signal Processing Hardware and writes the blocks to the Hardware
 - Start FFT – This operation instructs the Signal Processing Hardware to begin a Fast Fourier Transform
 - Read Data – This operation obtains data from the File
 - Write Data – This operation sends data to the File
 - Validate Parameters – This operation examines the parameters and determines if they will cause acceptable operation of the Hardware

- **Signal Sample**

- Create – This operation creates the actual pairs of sample number and signal values.

- **Time-Domain Signal**

- FFT – This operation transforms the Time-Domain Signal into a Frequency-Domain Signal.

- **User Interface**

- Initialize – This operation performs all the functions necessary to display the initial User Interface
- Display Graph – This operation receives the formatted data from the Discrete Digital Signal and sends it to the Display
- Display Interface – This operation formats the various portions of the User Interface and sends the information to the Display
- Display File – This operation receives the formatted data from the File and sends it to the Display
- Prompt User with Choices – This operation presents choices to the user and receives the chosen option
- Prompt User for Data – This operation presents information about needed data to the user and receives the data from the user
- Validate Parameters – This operation examines the data supplied by the user and determines if it is acceptable

- Execute Operating System Command - This operation translates the user-supplied system command to the appropriate OS-9 command, then causes execution of that command
- Shutdown - This operation updates any open files, closes them, and removes the User Interface from the Display

4.3.2 User Interaction Style Definition. User Interaction styles were discussed in Chapter II. Until this point, the actual interactions with the user have been abstracted away and treated as atomic operations. It is now time to decide how the user interface will actually look and how the user will interact with it. Based on the discussion in Chapter II, the primary method of user interaction is through menus. These menus present lists of choices and display sub-menus when appropriate - that is, when further parameters are needed that can be delineated. In cases where numerical parameters are needed, a fill-in menu is displayed. This fill-in menu lists the parameter(s) needed along with the acceptable range of values for that parameter. In some other cases, alphabetic input may be required, such as when a filename is needed. In this case, suggested inputs are displayed when available. For example, a list of available files is displayed when an existing filename is needed.

The user also has the option of either directly selecting the desired menu choice, or typing a distinguishing letter to indicate the desired selection. This alternative selection method allows users to develop familiarity with often-used command sequences and to apply this knowledge to increase the speed with which the system is used.

The appearance of the User Interface itself is also critical to successful user interaction and system performance. The proposed User Interface scheme is shown in Figure 12. The screen is divided into multiple interaction areas, with each area specializing in a particular type of input/output. The main menu is continuously displayed along the top of the display as a constant reminder of system capabilities. The main portion of the screen is dedicated to display of the signals or files of interest. A smaller portion of the display is devoted to system messages and direct user interaction. Sub-menus from the main menu are "dropped down" from the main menu, and sub-menus from those are displayed to the side of the parent sub-menu. Data fill-in menus appear centered on the display to focus attention and disappear once the data is accepted. System error messages appear in a highlighted box which will also disappear when the error condition is cleared.

4.3.3 Solution Space (Lower-Level) Object Addition. Solution space, or lower-level, objects are those objects which do not arise naturally from analysis of the system, but which are added to facilitate interfaces between other objects or to add detail to existing objects. For the PSK system, these objects are added primarily to add detail to the User Interface object and to provide a clean interface to the various pieces of signal processing hardware. Other solution space objects include those used to interface with the operating system and to provide for file storage. For the sake of brevity, the objects, attributes, and operations are all described together. They were, however, developed using the same procedure used for the development of the analysis-level objects: the objects were identified, then appropriate attributes were identified. Operations were identified based on filling in the details left out during the analysis process, and by developing abbreviated scenarios

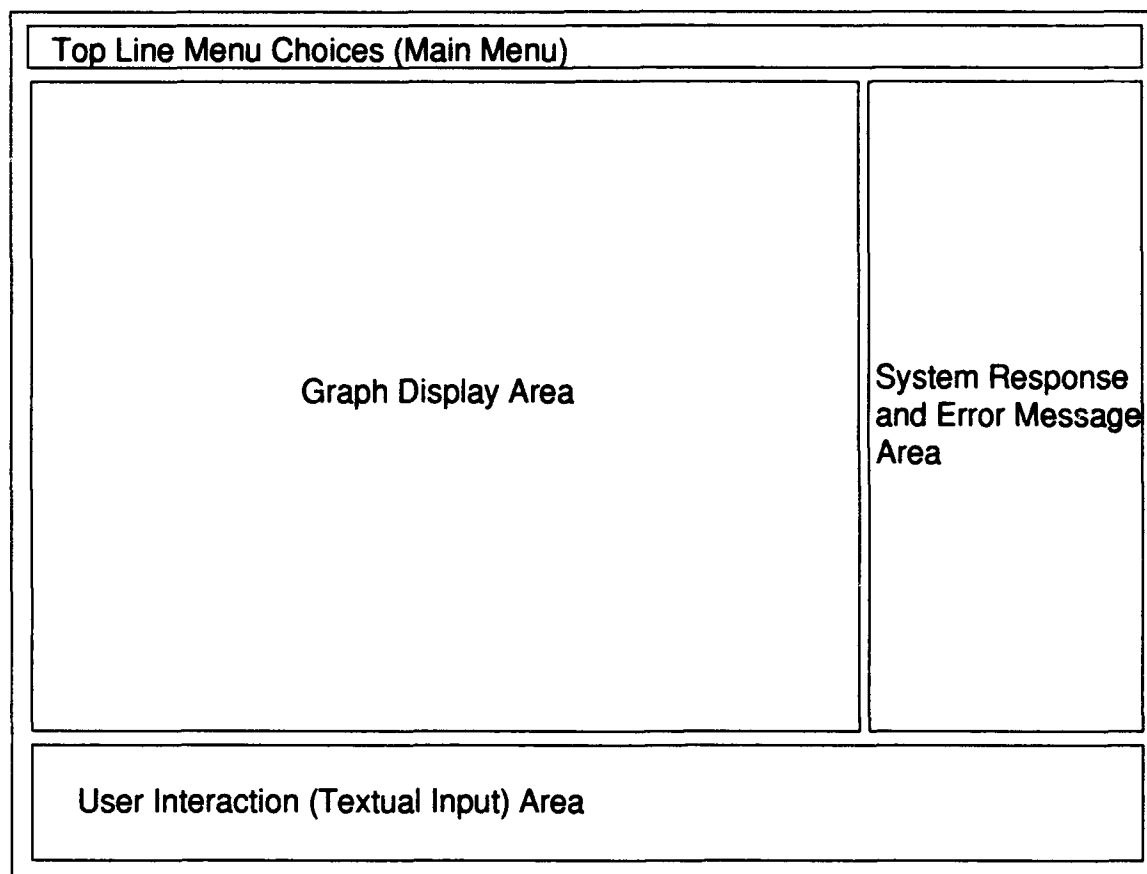


Figure 12. User Interface Scheme

and event lists. This abbreviated development cycle for the lower-level objects also resulted in changes to the dynamic and functional models, primarily by virtue of addition of extra states and processes. These changes are documented in the models described at the end of this section.

4.3.3.1 Object Identification. The objects added to the software system object diagram are named and described, in alphabetical order, as follows:

- **a66540 Interface** – The a66540 Interface object was added to encapsulate the operations needed for low-level interface to the a66540 FFT board and provide hardware-independent access to the facilities of the a66540.
- **Choice Items** – The Choice Item object represents a particular choice item on a Menu.
- **Data Fill-In Box** – The Data Fill-In Box object is a type of menu which supplies the user with needed data types, acceptable data values, and allows the user to enter the requested values in the fill-in blank contained in the box.
- **File Interface** – The File Interface object was added to provide the operations necessary to manipulate files.
- **Horizontal Bar** – The Horizontal Bar object is a particular type of menu which is constantly displayed, has a particular set of Choice Items, and appears as a horizontal line on the Display.
- **IV-3272 Interface** – The IV-3272 Interface object was added to encapsulate the operations needed for low-level interface to the IV-3272 Data transporter and provide hardware-independent access to the facilities of the IV-3272.

- **Menu** – The Menu object represents a list of choice alternatives, each of which is associated with a specific action.
- **Operating System Interface** – The Operating System Interface object was added to encapsulate the operations needed to convert user system-related commands into the syntax needed by OS-9.
- **Pop-Up Box** – The Pop-Up Box object is a particular type of menu which appears as a result of either a Choice Item being selected either from the Horizontal Bar or another Pop-Up Box.
- **Window** – The Window object represents the independent display areas provided by the User Interface.

4.3.3.2 New Association Identification. Few additional associations were created as a result of adding interface objects. The User Interface object has two new “controls” associations, one each for the Operating System Interface and File Interface objects. The User Interface is now defined to be made up of one or more Window objects and one or more Menu objects. The Menu object is defined to consist of one or more Choice Item objects, and has two concrete subclasses: the Horizontal Bar or Pop-Up Box objects. Finally, the Horizontal Bar class is associated with zero or more Pop-Up Box or Data Fill-In objects; each Pop-Up Box can be associated with zero or more Pop-Up Box objects or zero or more Data Fill-In objects. This last set of associations represents the notion that a top line menu choice may or may not be associated with a menu displaying further choices, and that further choices may be nested.

4.3.3.3 New Attribute Identification. The new objects also have attributes, which are listed by object class as follows:

- **a66540 Interface** – The a66540 Interface inherits its attributes from the Signal Processing Hardware object.
- **Choice Items** – The Choice Item has one attribute: Text String, which is the textual description corresponding to the function that will be executed when the Choice Item is selected
- **Data Fill-In Box** – The Data Fill-In Box inherits its attributes from the Menu object.
- **File Interface** – The File Interface object has five attributes: Name, which is the actual file name used by the system; Directory, the system directory in which the file is found; Permissions, the access permissions to the file as defined by OS-9; Type, the kind (ASCII or binary) of file; and Size, the number of bytes in the file.
- **Horizontal Bar** – The Horizontal Bar inherits its attributes from the Menu object.
- **IV-3272 Interface** – The IV-3272 Interface inherits its attributes from the Signal Processing Hardware object.
- **Menu** – The Menu has three attributes: Location, which is the set of four (x,y) pixel coordinates occupied by the edges of the Menu on the Display; Color, which is the background color of the Menu; Number of Items, which is the quantity of Choice Items displayed on the Menu.

- Operating System Interface – The Operating System Interface has two attributes: Current Data Directory and Current Execution Directory. These attributes are as defined in the OS-9 environment.
- Pop-Up Box – The Pop-Up Box inherits attributes from the Menu object, and adds one of its own: Highlight Color, which is the border color used to surround the Pop-Up Box upon creation to highlight it.
- Window – The Window has two attributes: Location, which is the set of four (x,y) pixel coordinates occupied by the edges of the Window on the Display; and Items Displayed, which is a set consisting of either Menu subclass objects, Signal Sample objects, or Discrete Digital Signal objects.

4.3.3.4 New Operation Identification. The operations on the new objects were identified and developed by examining the system events and processes in which the new object classes would be involved. These operations are listed by applicable object class as follows:

- a66540 Interface – This object inherits some operations from the superclass Signal Processing Hardware, and also includes the following:
 - Load Configuration – Using the configuration data supplied, this operation writes the configuration data to the a66540 and loads the appropriate program routines to the a66540.
 - Start FFT – This operation executes the command necessary to start the a66540.

- Stop FFT – This operation executes the command necessary to stop the a66540.
- Choice Items
 - Create – This operation writes the text specified to the proper location on the specified Menu.
 - Execute Function – This operation invokes the necessary operations on other objects to perform the requested service. It is activated by a mouse click when the pointer is over the Choice Item, or when the designated key is pressed.
- Data Fill-In Box– This object inherits its operations from the superclass Menu, and adds two more: Prompt User and Get Data.
 - Prompt User – This operation displays a particular data item needed and the acceptable range or ranges of values required for it.
 - Get Data – This operation reads the data supplied by the user and returns it to the calling object.
- File Interface
 - Create – This operation executes the necessary OS-9 commands to create a new file.
 - Read – This operation executes the necessary OS-9 commands to open a file, then reads the signal data from the file and sends it to the requesting object.
 - Write – This operation executes the necessary OS-9 commands to open a file, then writes signal data to the file in a predetermined format.

- Delete – This operation executes the necessary OS-9 commands to delete a file.
 - Display – This operation executes the necessary OS-9 commands to open a file, then formats the data and sends it to the Display as a two-dimensional graph.
 - List – This operation executes the necessary OS-9 commands to open a file, then formats the data and sends it to the Display as a list of discrete pairs of Signal Sample number values.
 - Copy – This operation executes the necessary OS-9 commands to copy a file.
 - Change Permissions – This operation executes the necessary OS-9 commands to change the permissions (read, write, and execute for the file's owner, other users, or both) on a file.
- Horizontal Bar – This object inherits its operations from the superclass Menu.
 - IV-3272 Interface – This object inherits some operations from the superclass Signal Processing Hardware, and also includes the following:
 - Load Configuration – Using the configuration data supplied, this operation creates the Transfer Parameter Blocks (TPB) that the IV-3272 needs to operate, and loads them into the IV-3272.
 - Load DSP Program – Using the configuration data supplied, this operation also creates and loads a TPB, but also loads TMS320C25 object code into the IV-3272.
 - Start Transfer – This operation executes the command to start the data transfer by the IV-3272.

- Stop Transfer – This operation executes the command to stop the data transfer by the IV-3272.

- Menu

- Create – This operation executes the necessary GFM/GSL function calls to create a new menu box on the Display.
- Remove – This operation executes the necessary GFM/GSL function calls to delete the specified menu from the Display.

- Operating System Interface

- Display Directory – This operation executes the necessary OS-9 commands to display the file names and sizes in either the current data or execution directory, whichever is specified.
- Change Directory – This operation executes the necessary OS-9 commands to change the current data or execution (as specified) directory.
- Create Directory – This operation executes the necessary OS-9 commands to create a new directory whose name is specified.
- Delete Directory – This operation executes the necessary OS-9 commands to remove the specified directory. The user must have the necessary permission.
- Show Current Directory – This operation executes the necessary OS-9 commands to display the pathname of the current data or execution directory, as specified.

- Display Date and Time - This operation executes the necessary OS-9 commands to display the current real time as maintained by the IV-3206 real time clock.
- Display Free Disk Space - This operation executes the necessary OS-9 commands to display the amount of available disk space on the specified drive.
- Display Free RAM Space - This operation executes the necessary OS-9 commands to display the amount of available system RAM space, with base addresses.
- Get Command Help - This operation executes the necessary OS-9 commands to display available system help on listed commands.
- Transfer File To or From System - This operation executes the necessary OS-9 and shell commands to transfer an ASCII file to or from an external system.
- Pop-Up Box- This object inherits its operations from the superclass Menu.
- Window
 - Create - This operation executes the necessary GFM/GSL function calls to create a new window on the Display.
 - Remove - This operation executes the necessary GFM/GSL function calls to delete the specified window from the Display.

4.3.4 Algorithm and Data Structures Definition. In this portion of the design process, the models are translated into data structures and procedures which can be translated to C source code. In many cases, the translation from the operation description was relatively straightforward; in others, research through the hardware technical manuals

was required. Some algorithms were partially reverse-engineered through examination of existing source code, and are marked as such. In any event, proper algorithm design has a critical effect on system performance. A poorly-designed or inefficient algorithm will slow the system and cause user frustration. Rumbaugh warns, however, that clarity and correctness should not be sacrificed purely for speed, and that the correct basic algorithm should be used prior to making minor tweaks for performance. With these considerations in mind, the algorithms were designed. All the algorithms are listed, ordered by object/operation, in Appendix B.

The selection of appropriate data structures goes hand-in-hand with the design of the algorithms. Poorly-chosen data structures force the design of poorly performing algorithms. Fortunately, only one object (along with its subclasses) is responsible for data storage. That object is the Discrete Digital Signal class, which is responsible for storing instances of its two concrete subclasses Time- and Frequency-Domain Signal. All other attribute data in other classes can be stored as simple variables. The Signals, however, are made up of multiple instances of Signal Samples which must be processed as a group. Because of the serial nature in which the samples are generated, the known size of sampled signals, and the need to access selected subsections of the Signal, an array is the most logical choice as the data structure to store them. Stacks, queues, and linked lists were considered but discarded due to the need to access parts of the Signal; random access into these types of structures is difficult or at least computationally expensive.

4.3.5 Control Method Definition. Rumbaugh discusses three styles of control: maintaining state as a function of location within the sequence of instructions defined by

the program code, implementation of a state machine transition manager, or implementing each object as a concurrently-running task alongside the other objects in the system [12]. Maintaining state information as the current location within the sequence of program instructions by the use of a main procedure and calls to subprocedures is the "traditional" method of state control and is adequate for highly procedural implementations. Having a state transition manager requires creation of a separate object for that function and incurs the extra overhead of performing an operation within the state transition manager whenever a state change is needed. Using concurrently-running tasks requires either language or run-time environment support and a message-passing procedure for task synchronization and communication.

For the PSK system implementation, the traditional approach of maintaining state as a function of location within the code is used. The OS-9 environment does provide for concurrent execution, but the requirements as currently stated do not call for concurrent execution of multiple tasks. As system requirements change, that may not necessarily continue to be the case. Use of a state transition manager was considered, but no advantages could be seen to outweigh the extra overhead required. The traditional approach provides the quickest execution time, which is necessary to ensure proper real-time performance.

4.3.6 Updating the Analysis Models. After completing several iterations of the above steps, the analysis models were updated to reflect the system design. As before, there is an object model, a dynamic model, and a functional model. The object model now reflects the added low level and interface objects developed as a result of design decisions. The dynamic model now reflects the extra states traversed/occupied by the previous analy-

sis objects and the added objects. The functional model likewise has additional lower-level processes associated with the design decisions made while elaborating the other models.

4.3.6.1 Object Model. For clarity, the Object model has been split into two components: one component containing the top-level User Interface object and the hardware interface objects, shown in Figure 13, and another containing the component objects of the User Interface, shown in Figure 14. The notation and syntax are the same as in Figure 6. These figures represent the complete object model of the PSK User Interface system; implementation of these objects is discussed in Chapter V.

4.3.6.2 Dynamic Model. The Dynamic Model reflects the designed dynamic behavior of the objects considered to have behavior worth modeling. Seven such objects are shown: the top-level User Interface object (Figure 15), whose behavior is of primary interest to the user; the Real-Time Front End Interface object (Figure 16), whose behavior is not necessarily apparent to the user, but is critical to proper system operation; the IV-3272 and a66540 Interface objects (Figure 17), whose behavior is visible to the user through the validation of parameters operations; and the Horizontal Box (Figure 18), Pop-Up Box (Figure 19), and Data Fill-In Box objects (Figure 20), which the user directly sees and interacts with. These models also have the same syntax and notation as Figures 8, 9, and 10.

4.3.6.3 Functional Model. The Functional Model has been carried forward from the analysis phase with few changes. The data flow diagram is shown in Figure 21, and includes new validation functions reflecting the validation operations added to the Object

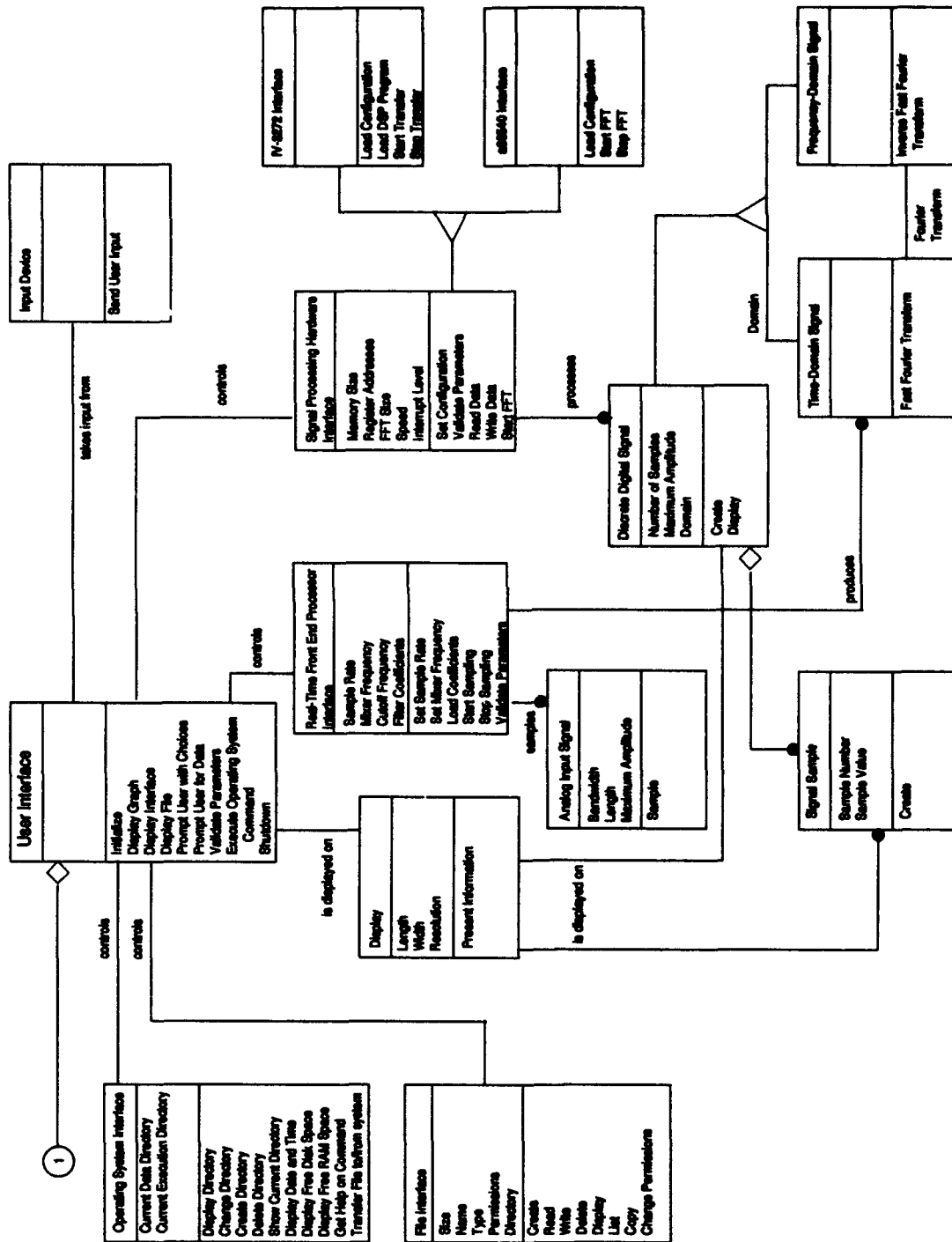


Figure 13. Software Design Object Model – Signal Processing and Interface Objects

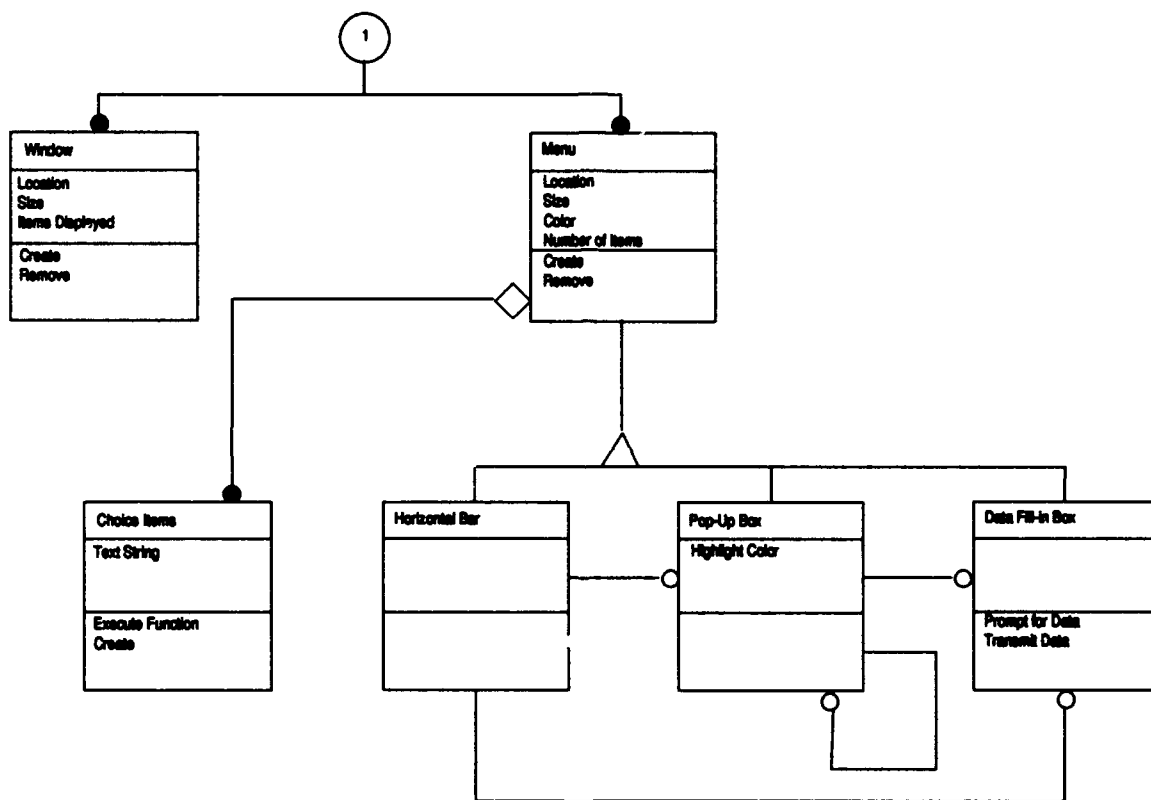


Figure 14. Software Design Object Model – User Interface Objects

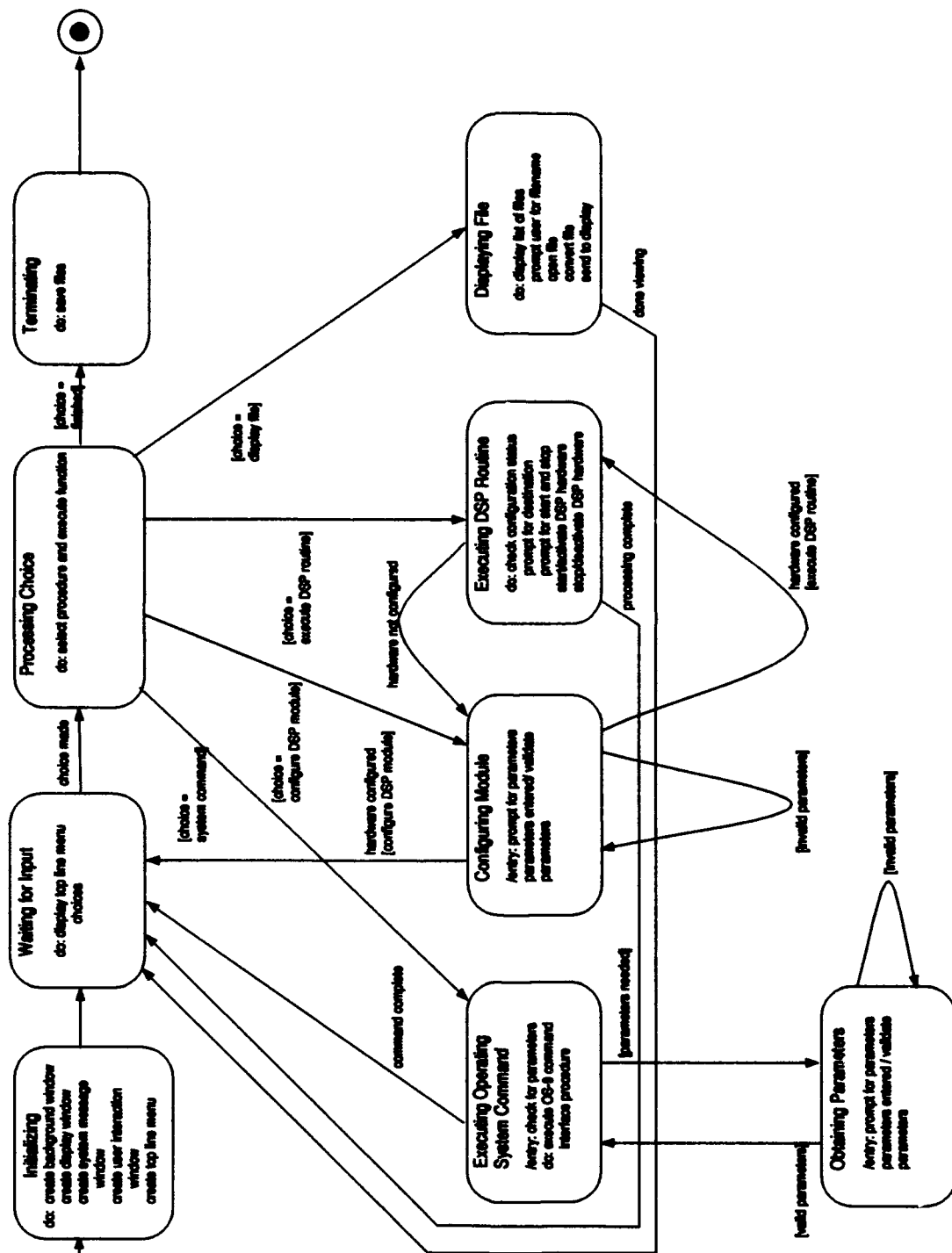


Figure 15. Software Design Dynamic Model – User Interface Object

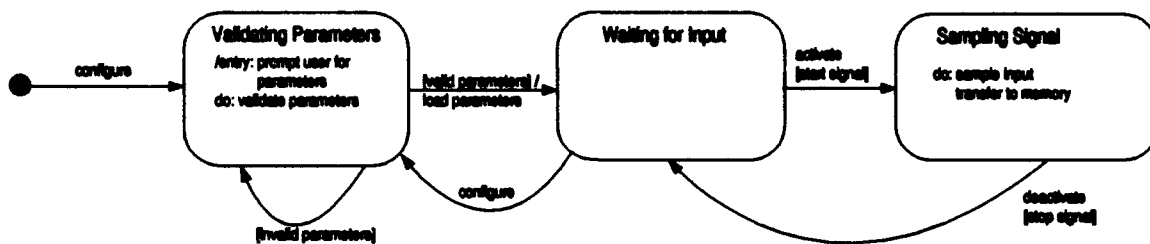


Figure 16. Software Design Dynamic Model – Real-Time Front End Processor Interface Object

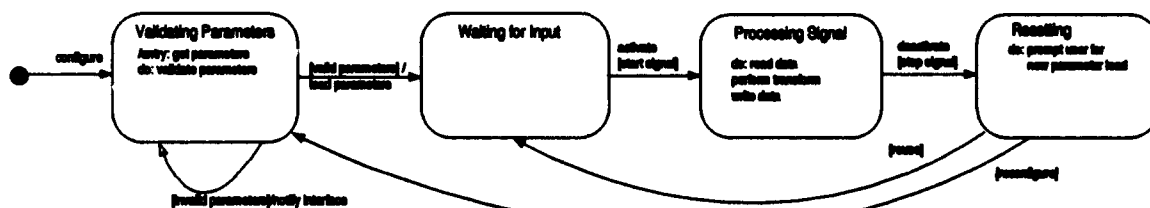


Figure 17. Software Design Dynamic Model – IV-3272 and a66540 Interface Objects

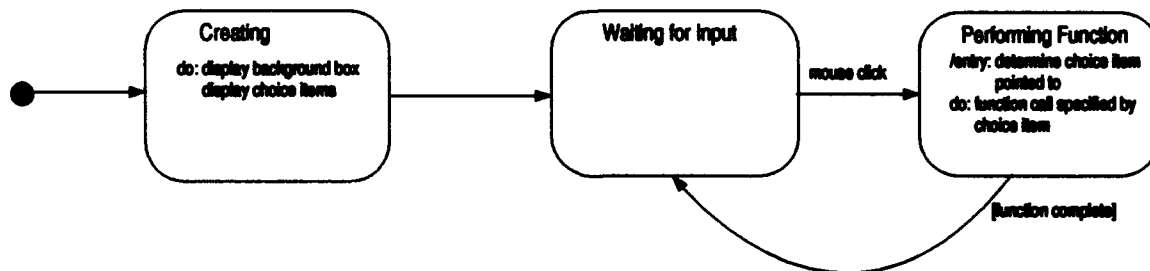


Figure 18. Software Design Dynamic Model – Horizontal Bar (Menu) Object

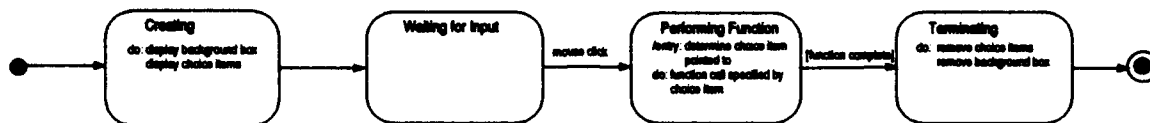


Figure 19. Software Design Dynamic Model – Pop-Up Box (Menu) Object

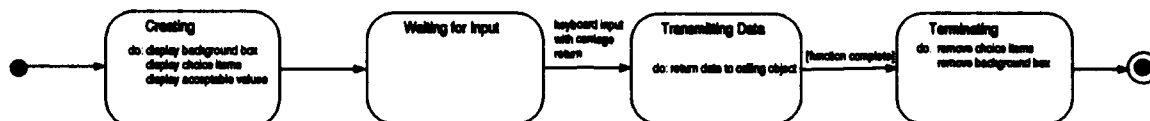


Figure 20. Software Design Dynamic Model – Data Fill-In (Menu) Object

Model. The other transforms remain the same as specified in the analysis model. The object operations and the algorithms used to implement them are detailed in Appendix B; essentially, the file conversion transforms are performed by the operations embedded in the File Interface Object, the validation transforms are performed by the validation operations in the hardware and Operating System Interface Objects, and the display graph transform is performed by operations in the User Interface Object. The central transforms of Sample and Filter and FFT are performed in hardware and managed by the appropriate Interface Objects.

4.4 Summary

This chapter discussed the analysis and design of the software required to implement the PSK System User Interface. It first discussed the analysis process used and the resulting models, using the notation developed by Rumbaugh. The requirements defined in Chapter III were analyzed, and an initial system object diagram was created. This

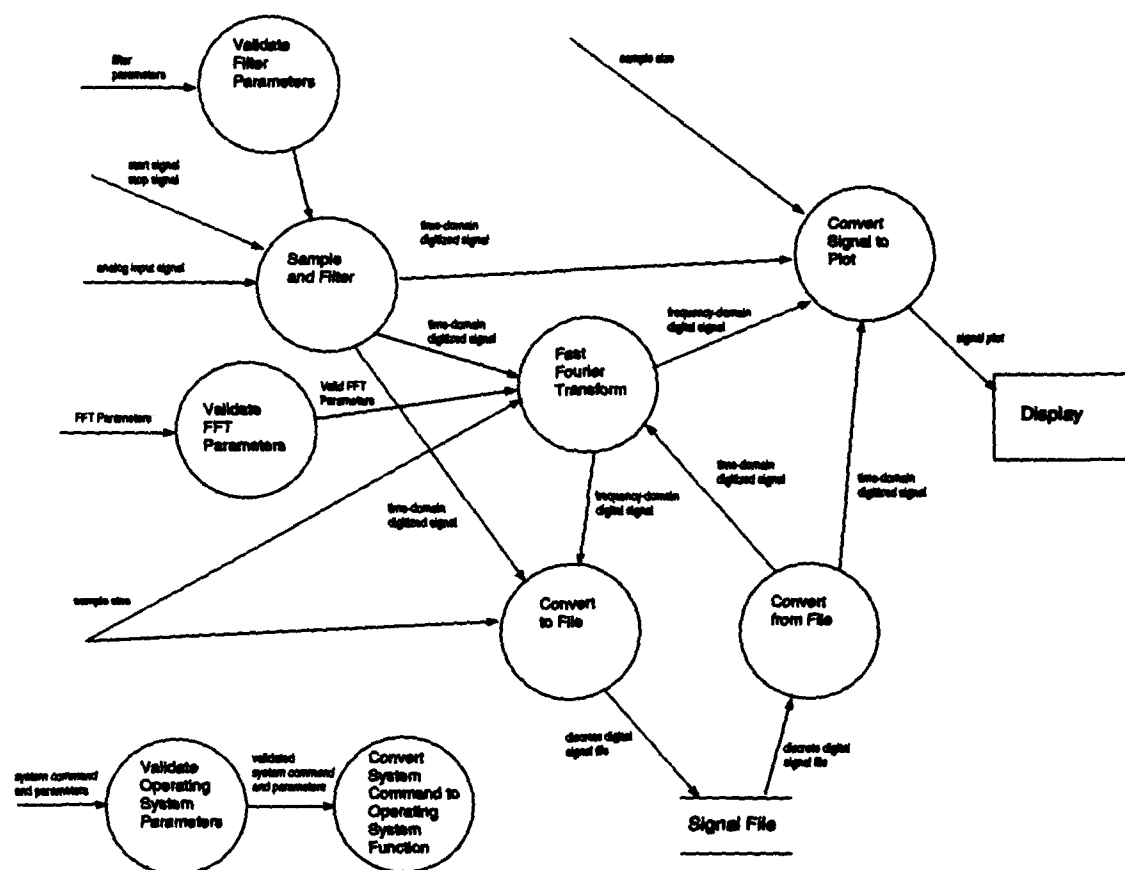


Figure 21. Software Design Functional Model - Data Flow Diagram

diagram was found to be too detailed for high-level analysis, so it was simplified to an object diagram containing only the essential object classes required for analysis. Object attributes and associations were then defined for these classes. The dynamic behavior of the system was then analyzed using typical interaction scenarios between the system and users. From the scenarios, system events were identified and the state behaviors of the identified object classes were defined based on those events. Inputs to and outputs from the system were then considered, and the necessary data transformations to produce the outputs from the inputs were created as part of the analysis functional model. A data flow diagram incorporating the data transformations was created.

The chapter then discussed the process used to turn the analysis models into design models usable for implementation. Operations on the analysis objects were defined by examination of the three models created during analysis. The user interaction style was elaborated using standard user interface design guidelines. Lower-level solution space objects were created to facilitate the interface between object classes or to model hardware interfaces. Attributes, associations, and operations of the new object classes were identified and described. Data structures, operation algorithms, and a method of control were defined. Finally, the updated design models and associated documentation were presented and discussed. A complete design document has been presented and is ready for implementation.

V. Software Implementation Plan

5.1 Introduction

This chapter outlines the process and details associated with an implementation of the user interface system design. Because time did not allow for a complete implementation of the user interface software system designed, these are recommended instructions for implementation. However, a limited prototype of the primary user interface menu was constructed using the recommended implementation techniques and is also outlined in this chapter.

5.2 Implementation Procedure

Implementation of the PSK System User Interface requires several steps. Because a true object-oriented language such as C++ is not available on the system, the design cannot be translated directly into the implementation language. Instead, the object-oriented concepts in the design must be mapped into a simulation of those concepts in (non-object-oriented) C. This C source code will have to be generated via one of three mechanisms: new code can be manually generated, new code can be automatically generated, or existing code can be modified and reused. In addition, file management and configuration control procedures need to be incorporated to ensure that the code generated is documented and controlled so that multiple revisions do not co-exist and cause confusion about which version is the most correct and current. The following sections discuss applicable techniques for each of these areas.

5.2.1 Translation into a Non-Object-Oriented Language. The logical choice for implementation of an object-oriented design is an object-oriented language such as C++. However, a C++ compiler is not available on the PSK system, so the implementation must be done in C. C does not inherently support object-oriented design features such as dynamic creation and deletion of objects, inheritance, and dynamic method resolution [12]. These features can be implemented in C by the use of `struct` type declarations, pointers to the `struct` variables, and pointers to functions.

Examination of the PSK System user interface design and available implementation tools reveals that this dynamic treatment of object classes is not necessary for a successful implementation. Most of the object classes in the design have only one instance (for example, the hardware interface objects). Providing dynamic allocation and deallocation for these objects is not necessary and only serves to introduce extra complexity into the implementation. Similarly, the RAVE Presentation Editor can automatically create source code for every instance of the Menu object subclasses needed to implement the user interface. In this case, these objects can be statically allocated to reduce the run-time complexity of the system. The system will thus have a minor increase in performance, at the cost of reduced flexibility. If the user interface must be reconfigured, the source code for the Menu objects will have to be regenerated using the Presentation Editor.

5.2.2 Code Generation and Editing. New code must be manually generated to implement those objects which cannot be automatically generated using the RAVE Presentation Editor or which cannot be implemented by adapting the code currently residing on the system. This code follows directly from the objects, attributes, and operations

documented in the design presented in Chapter IV and in the algorithms presented in Appendix B.

Creating new or editing existing code can be performed directly on the PSK system using the μ MACS editor. For developers thoroughly familiar with the `emacs` editor, this option may be the most viable. Other developers may want to use similar facilities available on the various AFIT networks. During system analysis, a connection to AFITNET was made to facilitate file transfers. Connecting to 129.92.3.189 via `telnet` results in the PSK system prompt, if the PSK system is operational. Furthermore, a direct serial connection was made between the system and the Z-248 microcomputer co-located on the lab bench. The Kermit communications program can be used to transfer files between either the PSK system and AFITNET or the PSK system and the Z-248. By transferring source files between these systems, the files may be created or edited with the desired editor and returned to the PSK system. Extreme caution is advised, however, to prevent multiple copies with slightly differing revisions from being retained on multiple systems. Currently, the source file structure of the PSK system is mirrored on the hawkeye system located in the Signal and Information Processing Laboratory located in room 2011.

5.2.3 User Interface Generation. Some of the User Interface objects can be directly generated using the RAVE Presentation Editor tool. The RAVE manuals, [34], [35], and [36], should be consulted directly for instructions on how to generate the code. The Presentation Editor generates, in addition to source code, configuration files which specify the location, size, color, and information contained in Menu objects. These configuration files have the same name as the C source file containing the source code for the menu

object, without the ".c" extension. Due to this automatic source code and configuration generation capability, the appearance and interaction style can be modified with less effort than would otherwise be required if the code had to be generated by hand. This modification procedure can be implemented by using the facilities of the Presentation Editor, and can be considered a form of prototyping – the appearance of the interface can be changed several times until an acceptable format is found. The user interface design guidelines referenced in Chapter II should still be followed.

5.2.4 Code Re-use. Some of the operations identified during the design phase can be implemented using source code supplied by the various hardware manufacturers, developed by SRL, or obtained from other sources. The source code supplied by the manufacturers is assumed to be correct, as it generally executed without errors during system analysis. The same cannot be said for the code supplied by SRL, as several of the source files provided by SRL either did not compile correctly or, if compiled, caused a bus error. A source for some of the file manipulation and lower-level user interaction routines is [39]. The operations and the corresponding source code packages or functions which can implement them (or be modified to do so) are listed in Appendix C.

5.2.5 Code Packaging. One of the features of the C programming language is the ability to compile and link functions contained in separate source files. Using separate source files for the various objects is one way to enforce differentiation between the various objects. In addition to using files as packages, a coding standard must be developed to enhance readability and maintainability of the generated code. The coding standard must address not only how the source code appears on the screen or printed page, but

also underlying stylistic concerns such as use of control structures, variable naming and initialization, behavior at boundary conditions, use of comments, and overall clarity. Two comprehensive sources for development of a coding standard are [40] and [41]; despite the latter's title, it contains many usable coding style concepts which are generally applicable for development in any programming language.

5.2.6 File Management and Configuration Control. The **make** system utility can be used to update the overall executable system whenever any of the component source files are changed. **Make** can also be used to generate a system file to indicate the time and date of the last full compilation. It cannot, however, keep track of multiple versions of the same source file and determine which is the most current. A separate recording system will be needed to maintain records of source code changes. This is especially important if a separate system is used for development and editing of source files.

5.3 System Administration

As currently configured, the PSK system operates in a quasi-single-user mode. Only two usernames are defined in the system password file, and both execute with superuser privileges. This configuration is acceptable for experimentation and development, but presents dangers for ordinary use. An ordinary signal processing user operating as a superuser could inadvertently delete important system files or otherwise disrupt the system. When the system software is fully developed, users should be "registered" with the system and given appropriate restrictions to prevent unauthorized access and modification to

system files. A separate directory could be created to hold user directories and data files, and system files could be protected with appropriate access permissions.

System backups are necessary, also. Currently, the tape drive does not function, and the system backups made to date have been on floppy disks. This backup method is acceptable in the short run, but the tape drive problem must be resolved in the long run to allow a more convenient and robust backup schedule.

5.4 Interface Prototyping

Using the techniques discussed above, a limited prototype of the PSK User Interface was constructed and placed in a new directory PROTOTYPE. The RAVE Presentation Editor was used to generate the single instance of the Horizontal Bar Menu object class (the Top-Level Menu) and five instances of Pop-Up Box Menu objects. These objects are static instantiations of the Top-Level Menu and the first level of submenus corresponding to the Choice Items on the Top-Level Menu. The Choice Items, with a few exceptions, make function calls to a function stub package to demonstrate/simulate the calls to operations which would be located in the interface objects described by the design. The functions not stubbed out are actual function calls located in a prototype Operating System Interface Object package, which are called by Choice Items contained in a Pop-Up Box which is created by the "Execute Operating System Function" Choice Item on the Top-Level Menu.

This prototype was constructed to demonstrate three concepts: the use of the RAVE Presentation Editor, the appearance of the User Interface itself, and the static creation of objects and operations. Using the Presentation Editor proved to be straightforward. The object instantiation process was guided by a series of menus which requested the text

items to appear on the menu, the functions which were to be called by each item, and the method by which the item would be selected. Once the items were instantiated, manual addition of function calls to each of the object source packages was required to link the Pop-Up Box Menu object creation operations with the Top-Level Menu calling functions. This process was also straightforward due to the automatic function prototyping performed by the Presentation Editor source code generator.

Once the source code for the User Interface prototype was created, it was compiled and run so the appearance of the interface could be evaluated. Executing the compiled code demonstrated that the User Interface appeared on the display exactly as designed and created: the Top-Level Menu appeared at the top of the display, and each of the sub-menus appeared just below the Top-Level Menu Choice Item with which they corresponded. Activating the sub-menu Choice Items resulted in the appropriate operation/function stub being executed, as evidenced by a one-line message appearing on the system console device.

The portion of the User Interface implemented appeared and behaved just as expected and designed, with one minor exception: after a certain number of Choice Item Executions, the interface would suspend execution. It appears that the creation of the sub-menu objects is not occurring exactly as anticipated. Furthermore, the Presentation Editor hung the system in a similar manner during initial object editing. Time did not permit further investigation of this phenomenon; however, it appears that in general the static allocation of the objects is sound. The interface was initialized correctly in all execution runs attempted, and the sub-menus did work correctly for a finite number of executions.

5.5 *Summary*

This chapter discussed implementation of the software interface system designed in this thesis. The chapter first discussed the techniques required to translate an object-oriented design into a non-object-oriented language and the tradeoffs resulting from that translation. The particular techniques available on the PSK system for the generation and modification of source code were then discussed, focusing on the use of the RAVE tools for generation of most of the user interface objects. Re-use of existing system code was briefly discussed, followed by a discussion of code packaging and coding style, which highlighted the use of good packaging and style to reinforce the object-oriented design concepts. The chapter then discussed the need for proper file management and configuration control to assure an orderly development. System administration was discussed, highlighting the need for system backups and proper identification of users for security reasons. Finally, a prototype implementation of the essential User Interface object was discussed, focusing on the method used to construct the objects, their appearance on the display, and the underlying function calls to interface objects. This chapter laid the groundwork for the implementation of the design presented in Chapter IV.

VI. Conclusion and Recommendations

6.1 Introduction

This chapter outlines the totality of actions taken during this thesis effort and provides guidance for future work on the PSK system. It first summarizes the development effort, focusing on the original objectives and how well they were met. It then discusses several recommendations for future work on the system in relative order of importance to successful system integration. Finally, the chapter discusses lessons learned about the system during the course of the development cycle and about the development cycle itself.

6.2 Summary

As outlined in Chapter I, the stated objectives of this thesis were to design a series of software routines and modules that allow an expert user to activate the system, select the analysis desired, and obtain the desired results without having to write any new software. To that end, a complete system and requirements analysis was performed to establish the baseline for the software system development. This initial analysis also spun off a complete system technical description which was lacking until now. The results of the requirements analysis were then used to develop initial object, dynamic, and functional models of the software system using Rumbaugh's object-oriented analysis techniques. Relevant object classes were identified and object class attributes and associations were determined. User interaction scenarios were developed, which were then used to develop the state transition diagrams for the various object classes. System inputs and outputs were identified and data transformations were developed.

As analysis was being concluded, the software system design was starting to be developed. Operations on the analysis objects were determined and a particular user interaction style was chosen. Solution-space object classes along with appropriate attributes, associations, and operations were added to the analysis models. Algorithms to implement the object class operations and an appropriate data structure were developed. The overall method of control was chosen and the analysis models were updated to reflect to addition of design details.

Time limitations prevented a full implementation of the developed design. However, an implementation plan was developed which addressed the translation of the design into a non-object-oriented language. The plan also addressed code generation concerns and system administration details. Furthermore, a limited prototype was constructed which successfully demonstrated part of the design as well as the implementation concepts presented.

Implementation of the full design as presented in this document should prove that the stated objectives were in fact met. Every requirement in the original requirements list can be mapped to one or more object operations which implement that requirement. Furthermore, the design has been developed in such a way as to enhance not only the ease of initial implementation, but future modifications and enhancements as well.

6.3 Recommendations

Much work remains to be done to make the PSK system completely usable for actual signal processing work. Both the hardware and software need to be modified to obtain a

fully-functional system. These actions are listed in perceived order of their importance in the following sections.

6.3.1 Buffer Memory. The first recommendation is to complete the design and implementation of the buffer memory. The PSK system will never fulfill its real-time processing function until the buffer memory is fully installed and tested. This will complete the real-time data path through the system and allow complete testing of the software designed in this thesis once it is implemented. One bank of memory is yet to be installed on the appropriate board. Also, the buffer memory as currently designed and partially implemented only transmits 10 of the 16 bits of information on each of the I and Q channels. This omission of the least significant six bits of information introduces quantization noise into the sampled signal by effectively reducing the number of quantization levels from 65536 to 1024.

6.3.2 Software Implementation. The second recommendation is to implement the software design that was developed in this thesis. Implementation of the software design documented in this thesis is absolutely essential to effective use of the PSK system. This work could be done in concert with the completion of the buffer memory, since most of the software does not require the buffer memory to be in place in order to be tested. The software should be implemented as specified by the design and the implementation guidelines specified in Chapter V.

Following an initial implementation, the next step should be to obtain a C++ compiler so that the object-oriented features of the analysis and design may be more fully

realized. Also, this will open up more possibilities for extensions to the design by allowing easier modifications and enhancements.

Another software development cycle issue which must be addressed is the creation of assembly-language code for the TMS320C25 DSP processor located on the IV-3272 Data Transporter. Unless code is written for that module, the IV-3272 will not be able to perform any DSP functions.

6.3.3 System Development Environment. The final recommendation is to correct the deficiencies in the PSK system environment. In fact, some of the problems in the system environment can and should be corrected before the bulk of the software implementation occurs. Otherwise, the implementor will spend excess time tracing system-generated problems during compilation. The following deficiencies must eventually be corrected, and the first three should be addressed concurrently with system development:

- **Directory Structure** – The file directory structure must be simplified to group together similar software modules. System utilities should be placed together (both in source and executable form). Hardware interface routines likewise should be grouped hierarchically. Relocatable object modules should likewise be grouped with the source files which generated them.
- **Makefiles** – The system makefiles are not standardized and in some cases do not take advantage of the optimizations available for the 68030 microprocessor. “Not standardized” means that two or more makefiles will use different macro expansions to accomplish the same work; where possible, they should be rewritten to use the same macros and flags so that their operation can be easily understood. Moving the

directory structure around as noted above will contribute to the ease of doing these revisions.

- **Initialization** – This problem area has several facets. First, there are multiple versions of the system initialization files in existence; the non-current versions need to be archived or otherwise flagged. Second, the paths loaded at initialization must be revised to reflect the changes made in the directory structure. Third, the system should be configured to automatically boot into OS-9 without stopping at the IV-3206 boot monitor – this requires a modification to the boot monitor and change to one strap on the IV-3206 board.
- **Tape Backup** – The device descriptors and driver for the tape backup need to be compiled and linked to make the tape backup drive available. Backups are now being made using floppy disks, which is inconvenient at best.
- **Addition of User Directories** – Once the system is otherwise fully operational with the designed software, the password file and file directories should be modified to allow different users to store their files separately. Currently, all users login with the same username and password. This method is acceptable for casual use, but is insecure for general use with multiple users.

6.4 *Lessons Learned*

By far, the most important lesson learned during this study was the need for intelligent planning and realistic estimates of the time required to complete certain tasks. This thesis appeared at first to be a straightforward software development effort. However,

system design and re-engineering issues soon appeared to be at least equal to, if not more formidable than, the software issues to comprehend. Too much time had to be devoted to background research and understanding, leaving the actual software analysis and design to be worked double-time at the end of the thesis cycle. The next developer should use this thesis as a starting point to understand the features, problems, and idiosyncrasies of the system.

Another lesson learned was the need to get hands-on experience with the system in order to gain a full understanding of its capabilities and limitations. Reading manuals and hardware descriptions is fine, but nothing replaces actually powering up the system and seeing what the Makefiles do, or observing the initialization sequence, or having a vendor-supplied utility crash the system with a bus error.

6.5 Conclusion

At the beginning of this thesis cycle, the PSK system was a largely unknown quantity. Clearly, it was a system with great, but untapped, potential. This thesis study has laid the groundwork for development of the system to its full capabilities and should be used as the starting point for future work with the system.

Appendix A. System Behavior Scenarios

In Chapter IV, part of the development of the analysis dynamic model included the development of scenarios intended to reflect system behavior during both normal and erroneous user interaction sequences. These scenarios are listed here for completeness.

- Scenario 1 – User wants to execute an operating system command with or without parameters

1. User selects “Execute operating system command” from list of available commands
2. System responds with list of choices
3. User selects desired action
4. System asks for any necessary parameters, which are determined by the system based on the action desired
5. User supplies parameters
6. If parameters supplied are incorrect, the system responds with help list of correct parameter values or characteristics
7. User supplies correct parameters (if not, repeat above item)
8. System executes command and returns to main list

- Scenario 2 – User wants to display a signal file

1. User selects “Display signal file” from main list
2. System responds with filename query

3. User supplies filename
 4. If the filename does not match with an available file, the system displays File not Found message and list of available files
 5. User supplies correct filename (if not, repeat above item)
 6. System opens file and displays on main display
 7. User indicates completion of view
 8. System clears screen and returns to main list
- Scenario 3 – User wants to configure a processor board
 1. User selects “Configure processor board” from main list
 2. System displays list of available boards for configuration
 3. User selects board
 4. System displays fill-in list of parameters needed
 5. User enters parameters
 6. System validates parameters; if acceptable, writes them to the board, and returns to main list; if not, displays list of acceptable ranges for parameters found faulty
 7. User enters parameters
 8. System repeats validation; if acceptable, writes to board and returns; if not, above sequence is repeated
 - Scenario 4a – User wants to process a signal file (processor boards previously configured)

1. User selects "Process signal file" from main list
 2. System displays list of available processing techniques and associated hardware
 3. User enters choice
 4. System verifies board configuration and prompts for destination: display, file, or both
 5. User enters choice
 6. System prompts for source and destination filenames if file option is chosen
 7. User enters filenames
 8. System verifies existence of source filename and absence of destination (i.e., no overwrite) filename; starts processing; returns to main list at end of processing
- Scenario 4b – User wants to process a signal file (processor boards not previously configured)

1. User selects "Process signal file" from main list
2. System displays list of available processing techniques and associated hardware
3. User enters choice
4. System verifies board configuration; configuration not found, so second step of either scenario 3 is entered; once configured, prompts for destination: display, file, or both
5. User enters choice
6. System prompts for source and destination filenames if file option is chosen

7. User enters filenames
 8. System verifies existence of source filename and absence of destination (i.e., no overwrite) filename; starts processing; returns to main list at end of processing
- Scenario 5 – User wants to capture incoming analog signal
 1. User selects “Capture real-time signal” from main list
 2. System checks configuration of real-time front end processor. If configured, go on; if not, execute interaction sequence as in scenario 3a/b.
 3. System displays list of acceptable destinations: display, file, or both
 4. User chooses destination (if file has been selected, a series of interactions as in steps 2-6 of scenarios 2 is executed)
 5. System prompts for start and stop time of signal
 6. User enters times
 7. System validates times against real-time clock (unacceptable times are negative, in the past, or stop time prior to start time)
 8. System waits until start time; when reached, starts capturing data and writing to file; when stop time is reached or user commands a halt, stop capturing data, close file and return to main user list.
 - Scenario 6 – User wants to capture incoming analog signal and process in real time
 1. User selects “Process real-time signal” from main list
 2. System prompts for processing technique and associated hardware

3. User chooses hardware
 4. System checks configuration status of hardware; If configured, go on; if not, execute interaction sequence as in scenario 3.
 5. System responds with list of acceptable destinations: display, file, or both
 6. User chooses destination (if file has been selected, a series of interactions as in steps 2-6 of scenarios 2 is executed)
 7. System checks configuration of real-time front end processor. If configured, go on; if not, execute interaction sequence as in scenario 3.
 8. System waits for user command to start sampling; when received, starts capturing data and writing to file; when user commands a halt, stop capturing data, close file and return to main user list.
- Scenario 7 – User is finished using the system
 1. User selects “Shutdown system” from main list
 2. System responds with confirmation request
 3. User confirms shutdown
 4. System checks for memory images corresponding to processed signals; if found, user is queried if saving image to a file is desired
 5. User indicates save preference; if save is desired, an interaction the same as in scenarios 2a/b is performed to determine filename
 6. System writes signal memory images, if any, to the file selected
 7. System exits procedure and returns control to the VT-340

Appendix B. Operation Algorithm Definitions

This appendix details the algorithms needed to implement the operations on each object. They are grouped by object and are written in structured English, psuedocode, or both, whichever is appropriate to document the algorithm. Low-level system or function calls are not necessarily shown with the exact syntax needed, but are identified in such a way that translation should be straightforward. In the algorithm descriptions, calls to an object operation are shown in the form Object-Name.Operation-Name or, when inheritance is involved, Object-Name(SubObject-Name).Operation-Name. Operations included in abstract object classes are shown with that object class, but would be instantiated with the concrete subclass.

B.1 User Interface

Initialize:

Invoke the Window.Create operation to create the background window

Invoke the Menu(Horizontal Bar).Create operation to create the Top-Level Menu

Invoke the Window.Create operation to create the Graph Display Window

Invoke the Window.Create operation to create the System Response and Error Messages Window

Invoke the Window.Create operation to create the User Interaction Window

Set the Hardware Configured flags to "not configured"

Display Graph:

Invoke the Discrete Digital Signal.Display operation

Based on the number of samples included in the Discrete Digital Signal and the maximum amplitude of the samples, draw a labeled set of axes in the Graph Display Window

Invoke the Menu(Pop-Up Box).Create operation to create an interaction menu for the user to indicate when viewing of the graph is completed.

When user indicates that viewing is complete (by a value passed back by the Choice Item.Execute Function operation), blank the Window area occupied by the graph drawn

Display Interface:

This operation is handled in hardware by the Vigna MMI-250 board.

Display File:

Open the indicated File

Invoke the File.Read operation starting at the beginning of the file

Write the ASCII Sample number, magnitude, and phase values to the System Error Messages and Interaction Window, stopping every 24th line and prompting the user with a Pop-Up Box to continue displaying the samples

Prompt User with Choices:

When the mouse pointer is over one of the Choice Items in the Top Level Menu and the left mouse button is clicked, or when the appropriate key is depressed, invoke the Menu(Pop-Up Box).Create operation.

Prompt User for Data:

When the mouse pointer is over one of the Choice Items in the Top Level Menu and the left mouse button is clicked, or when the appropriate key is depressed, invoke the Menu(Data Fill-In Box).Create operation.

Validate Parameters:

Using the function name returned by the Choice Item.Execute Function operation, invoke the validate parameters operation on either the Real Time Front End Interface object or the Signal Processing Hardware object

Execute Operating System Command:

This operation is performed by the operations in the Operating System Interface object

Shutdown:

Invoke the Window.Remove operation to remove the User Interaction Window

Invoke the Window.Remove operation to remove the System Response and Error Messages Window

Invoke the Window.Remove operation to remove the Graph Display Window

Invoke the Menu(Horizontal Bar).Remove operation to remove the Top-Level Menu

Invoke the Window.Remove operation to remove the background window

B.2 Operating System Interface**Display Directory:**

When the mouse pointer is over this Choice Item and the left mouse button is clicked or when the highlighted keystroke is made, execute the appropriate system call to obtain the filename contents of the current data directory and return the resulting text to the System Interaction Window.

Change Directory:

When the mouse pointer is over this Choice Item and the left mouse button is clicked or

when the highlighted keystroke is made, invoke the Menu(Data Fill-in).create operation to ask for the desired change.

When user indicates the desired directory (by a value passed back by the Choice Item.Execute Function operation), execute the appropriate system call to change the pathname of the current data directory.

Create Directory:

When the mouse pointer is over this Choice Item and the left mouse button is clicked or when the highlighted keystroke is made, invoke the Menu(Data Fill-in).create operation to ask for the desired directory name.

When user indicates the desired directory (by a value passed back by the Choice Item.Execute Function operation), execute the appropriate system call to create the pathname of the desired directory.

Delete Directory:

When the mouse pointer is over this Choice Item and the left mouse button is clicked or when the highlighted keystroke is made, invoke the Menu(Pop-Up Box).create operation to ask for confirmation of the deletion.

If the user indicates affirmative, (by a value passed back by the Choice Item.Execute Function operation), invoke the Menu(Data Fill-in).create operation to ask for the desired deletion.

When user indicates the desired directory (by a value passed back by the Choice Item.Execute Function operation), execute the appropriate system call to delete the indicated directory.

If the user indicates no deletion, abort the operation.

Show Current Directory:

When the mouse pointer is over this Choice Item and the left mouse button is clicked or when the highlighted keystroke is made, execute the appropriate system call to obtain the pathname of the current data and execution directories and return the result to the System Interaction Window.

Display Date and Time:

When the mouse pointer is over this Choice Item and the left mouse button is clicked or when the highlighted keystroke is made, execute the appropriate system call to obtain the current real system date and time and return the result to the System Interaction Window.

Display Free Disk Space:

When the mouse pointer is over this Choice Item and the left mouse button is clicked or when the highlighted keystroke is made, execute the appropriate system call to obtain the current amount of free space on the hard disk and return the result to the System Interaction Window.

Display Free RAM Space:

When the mouse pointer is over this Choice Item and the left mouse button is clicked or when the highlighted keystroke is made, execute the appropriate system call to obtain the amount of free RAM space and return the result to the System Interaction Window.

Get Help on Command:

When the mouse pointer is over this Choice Item and the left mouse button is clicked or when the highlighted keystroke is made, invoke the Menu(Pop-Up Box).Create operation, with Choice Items corresponding to the available functions and a text message instructing

the user to select a command

When the Choice Item.Execute Function operation is executed, invoke the Menu(Pop-Up Box).Create operation to display a box containing help text for the command and a Choice Item to remove the help box

When the Choice Item.Execute Function operation is executed, recursively call Menu.Remove to remove the Menu objects until only the Top Level Menu remains

Transfer File:

When the mouse pointer is over this Choice Item and the left mouse button is clicked or when the highlighted keystroke is made, invoke Menu(Pop-Up Box).Create to display a menu containing help text and two choices: Transfer File From PSK System, and Transfer File To PSK System

If Choice Item.Execute Function is Transfer File To PSK System, do the following:

Invoke Menu(Data Fill-In Box).Create to prompt for the directory and file name

Check to see if that file currently exists; if so, invoke Menu(Pop-Up Box).Create to display a choice of overwrite or no overwrite

If Choice Item.Execute Function is overwrite, continue on with process; if not (no overwrite), invoke Menu(Pop-Up Box).Remove to erase the latest Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates exit, invoke Menu.Remove until only the Top Level Menu remains

If overwrite is allowed, or if file does not currently exist on the PSK System, invoke Menu(Pop-Up Box).Create to display a menu with transfer instructions and a Choice Item to indicate the start of transfer

When Choice Item.Execute Function indicates the start of transfer, execute the system shell command "kermit -r" with filename

When transfer is finished, invoke Menu.Remove repeatedly until only the Top Line Menu remains

If Choice Item.Execute Function is Transfer File From PSK System, do the following:

Invoke Menu(Data Fill-In Box).Create to prompt for the directory and file name

Check to see if that file currently exists; if not, invoke Menu(Pop-Up Box).Create to display a warning and a Choice Item to indicate another try

When Choice Item.Execute Function indicates to try again, invoke Menu.Remove to remove the Pop-Up Box and return to the Data Fill-In Box

If the file does not currently exist on the PSK System, invoke Menu(Pop-Up Box).Create to display a menu with transfer instructions and a Choice Item to indicate the start of transfer

When Choice Item.Execute Function indicates the start of transfer, execute the system shell command "kermit -s" with filename

When transfer is finished, invoke Menu.Remove repeatedly until only the Top Line Menu remains

B.3 File Interface

Create:

Invoke Menu(Data Fill-In Box).Create to prompt for the directory and file name

Check to see if that file currently exists; if so, invoke Menu(Pop-Up Box).Create to display a choice of overwrite or no overwrite

If Choice Item.Execute Function is overwrite, continue on with process; if not (no overwrite), invoke Menu(Pop-Up Box).Remove to erase the latest Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates exit, invoke Menu.Remove until only the Top Level Menu remains

If the file does not currently exist on the PSK System, invoke Menu(Data Fill-In Box).Create to display a menu to obtain the size of the signal

Invoke File.Write the number of times specified by the signal size

When creation is finished, invoke Menu.Remove repeatedly until only the Top Line Menu remains

Read:

Invoke Menu(Data Fill-In Box).Create to prompt for the directory and file name

Check to see if that file currently exists; if not, invoke Menu(Pop-Up Box).Create to display a warning and a Choice Item to indicate another try

When Choice Item.Execute Function indicates to try again, invoke Menu.Remove to remove the Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates exit, invoke Menu.Remove until only the Top Level Menu remains

If file exists, create an array large enough to hold the samples in the file

Read sequentially through the file and load the array

At end of file, close it and invoke Menu.Remove

Write:

If called with user input, create an array and invoke Menu(Data Fill-In Box).Create to

prompt the user for signal values

When values are entered, write to array and invoke Menu.Remove to erase the data fill in box

Repeat data prompts until array is full

When array is full, open file and write to it

Invoke Menu.Remove to erase original data fill in box

If not called with user input, open file and write array to it

Delete:

Invoke Menu(Data Fill-In Box).Create to prompt for the directory and file name

Check to see if that file currently exists; if not, invoke Menu(Pop-Up Box).Create to display a warning and a Choice Item to indicate another try or exit

When Choice Item.Execute Function indicates to try again, invoke Menu.Remove to remove the Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates exit, invoke Menu.Remove until only the Top Level Menu remains

Execute system call to check permissions on file; if permissions allow delete, proceed, if not, display a warning in the System Error Messages window and invoke Menu.Remove to erase menus

Invoke Menu(Pop-Up Box).Create to display a warning and a Choice Item to indicate confirmation of delete

When Choice Item.Execute Function indicates go, execute system call to delete file

Invoke Menu.Remove to erase original data fill in box

Display:

Invoke Menu(Data Fill-In Box).Create to prompt for the directory and file name

Check to see if that file currently exists; if not, invoke Menu(Pop-Up Box).Create to display a warning and a Choice Item to indicate another try or exit

When Choice Item.Execute Function indicates to try again, invoke Menu.Remove to remove the Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates exit, invoke Menu.Remove until only the Top Level Menu remains

Invoke File.Read

Invoke User Interface.Display Graph

List:

This operation is the same as the User Interface.Display File operation and thus invokes it

Copy:

Invoke Menu(Data Fill-In Box).Create to prompt for the source directory and file name

Check to see if that file currently exists; if not, invoke Menu(Pop-Up Box).Create to display a warning and a Choice Item to indicate another try

When Choice Item.Execute Function indicates to try again, invoke Menu.Remove to remove the Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates exit, invoke Menu.Remove until only the Top Level Menu remains

Invoke Menu(Data Fill-In Box).Create to prompt for the target directory and file name

Check to see if that file currently exists; if so, invoke Menu(Pop-Up Box).Create to display a choice of overwrite or no overwrite

If Choice Item.Execute Function is overwrite, continue on with process; if not (no overwrite), invoke Menu(Pop-Up Box).Remove to erase the latest Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates to try again, invoke Menu.Remove to remove the Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates exit, invoke Menu.Remove until only the Top Level Menu remains

If the file does not currently exist on the PSK System or if overwrite is allowed, execute system call to copy file

Invoke Menu.Remove until only Top Level Menu remains

Change Permissions:

Invoke Menu(Data Fill-In Box).Create to prompt for the directory and file name

Check to see if that file currently exists; if not, invoke Menu(Pop-Up Box).Create to display a warning and a Choice Item to indicate another try

When Choice Item.Execute Function indicates to try again, invoke Menu.Remove to remove the Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates exit, invoke Menu.Remove until only the Top Level Menu remains

Execute system call to check permissions on file; if permissions allow modification, proceed, if not, display a warning in the System Error Messages window

Invoke Menu(Pop-Up Box).Create to display allowable changes

When Choice Item.Execute Function is received, execute system call to modify file per-

missions

Invoke Menu.Remove to erase original data fill in box

B.4 Real-Time Front End Processor Interface

Set Sample Rate:

Invoke Menu(Pop-Up Box).Create to display choice of internal or external clock

When Choice Item.Execute Function indicates internal clock rate, invoke Menu.Remove to remove the Pop-Up Box, set the external clock flag to false and clock frequency to 20 MHz

When Choice Item.Execute Function indicates external clock rate, invoke Menu(Data Fill-In Box).Create and Data Fill-In Box.Prompt User to prompt for the frequency with a limit of 20 MHz

Invoke Validate Parameters

If parameters are not acceptable, invoke Menu.Remove and repeat above step, otherwise proceed

Invoke Data Fill-In Box.Send Data to set the clock frequency and set external clock flag to true

Invoke Menu.Remove to erase original data fill in box

Set Mixer Frequency:

Invoke Menu(Data Fill-In Box).Create and Data Fill-In Box.Prompt User to prompt for the frequency with a limit of 10 MHz

Invoke Validate Parameters

If parameters are not acceptable, invoke Menu.Remove and repeat above step, otherwise

proceed

Invoke Data Fill-In Box.Send Data to set the mixer frequency

Invoke Menu.Remove to erase original data fill in box

Load Coefficients:

Using the sample clock frequency and mixer frequency, calculate 64 tap FIR low-pass filter coefficients

Iterate through filter chip VMEbus addresses and write coefficients so calculated to the addresses

Start Sampling:

Invoke Menu(Pop-Up Box).Create to display start and sample size

When Choice Item.Execute Function indicates start, create array of given size

Write samples to array until Stop Sampling is invoked or array limit is reached

Stop Sampling:

Invoke Menu(Pop-Up Box).Create to display stop signal

When Choice Item.Execute Function indicates stop, record position in array and save as Sample Size

Validate Parameters:

Compare data value input with allowable input range

If acceptable, return 1; if not, return 0

B.5 Signal Processing Hardware Interface

Set Configuration:

Invoke Menu(Pop-Up Box).Create to display allowable hardware choices

When Choice Item.Execute Function indicates choice, invoke either a66540 Interface.Load Configuration or IV-3272 Interface.Load Configuration depending on which hardware object is selected

Validate Parameters:

Compare the size of the array to the size of the desired FFT

If equal, return 1, if not, return 0

Read Data:

Invoke File.Read

Write Data:

Invoke Menu(Data Fill-In Box).Create to prompt for the directory and file name

Check to see if that file currently exists; if so, invoke Menu(Pop-Up Box).Create to display a choice of overwrite or no overwrite

If Choice Item.Execute Function is overwrite, continue on with process; if not (no overwrite), invoke Menu(Pop-Up Box).Remove to erase the latest Pop-Up Box and return to the Data Fill-In Box

When Choice Item.Execute Function indicates exit, invoke Menu.Remove until only the Top Level Menu remains

Invoke File.Write without user input

Start FFT:

Invoke Menu(Pop up Box).Create to display a pop up box which will signal start

When Choice Item.Execute Function indicates to start, invoke either a66540 Interface.Start
FFT or IV-3272 Interface.Start Transfer depending on which hardware object is selected

B.6 IV-3272 Interface**Load Configuration:**

Create Transfer Parameter Block (TPB)

Write TPB to IV-3272 FIFO Register

Set IV-3272 configured flag to "true"

Load DSP Program:

Create Transfer Parameter Block (TPB) with a \$8000 sink address and user code source
address

Write TPB to IV-3272 FIFO Register

Invoke Start Transfer

Start Transfer:

Write "1" to IV-3272 Start/Stop Register

Stop Transfer:

Write "0" to IV-3272 Start/Stop Register

B.7 a66540 Interface

Load Configuration:

Invoke Menu(Data Fill-In Box).Create to display an input box to get the number of points in the FFT desired

When Data Input Box.Send Data returns the value, invoke Validate Parameters

If parameters are not acceptable, invoke Menu.Remove to erase the data fill in box and repeat above; otherwise continue

Initialize the a66540 using function call

Load the a66540 program memory and control registers

Generate and load twiddle factors based on the size of the FFT

Set a66540 configured flag to "true"

Start FFT:

Invoke Signal Processing Hardware.Read Data

Load Array into a66540 from VMEbus memory

Initiate processing with a66540 function call

Invoke Menu(Pop up Box).Create to display a pop up box which will signal stop

When Choice Item.Execute Function indicates to stop, invoke Stop FFT and Menu.Remove to erase pop up box

Stop FFT:

Write array to VMEbus memory

Invoke Signal Processing Hardware.Write Data

Invoke File.Display

B.8 Discrete Digital Signal

Create:

Allocate memory to hold an array

Receive sample values from either the Real-Time Front End Processor (Signal Sample.Create operation) or from the File.Read operation, until the array is filled

Display:

Iterate over the array and return the sample number, sample magnitude, and sample phase to the invoking object

B.9 Signal Sample

Create:

This operation is performed in hardware by the Real-Time Front End Processor.

B.10 Time-Domain Signal

Fast Fourier Transform:

This operation is performed in hardware by the a66540 FFT board or the IV-3272 Data Transporter, if configured.

B.11 Frequency-Domain Signal

Inverse Fast Fourier Transform:

This operation is performed in hardware by the a66540 FFT board or the IV-3272 Data Transporter, if configured.

B.12 Window

Create:

Using the parameters (attributes) supplied by the invoking function, execute the system calls necessary to set aside the designated pixel area on the Display.

Remove:

Execute the system calls necessary to delete the reserved pixels from the display.

B.13 Menu

Create:

Using the parameters (attributes) supplied by the invoking function, execute the system calls necessary to set aside the designated pixel area on the Display and save the pixel image underneath the area to be occupied by the Menu.

Remove:

Execute the system calls necessary to delete the reserved pixels from the display and restore the saved image overwritten when the create operation was invoked.

B.14 Choice Item

Create:

Read the prompt text from an ASCII configuration file

Create a pointer to the function executed when the item is selected

Write the prompt text to the parent Menu

Execute Function:

When the mouse pointer is over the text string and the left mouse button is depressed or the highlighted key is depressed, initiate execution of the function pointed to and invoke the Menu.Remove operation

B.15 Data Input Box**Prompt User:**

Using parameters supplied by the invoking object, display the range of acceptable parameters

When the mouse pointer is inside the box, and an entry is made (as signified by a carriage return), compare the data entered to the input parameters

If acceptable, invoke the Send Data operation

If not acceptable, erase the input area, reposition the cursor, and display a warning message on the System Error and Interaction Window

Send Data:

Return the data entered to the invoking object and invoke the Remove operation

Appendix C. Source Code Sources

This appendix maps the functions called or referenced in Appendix B to the existing source code package or packages which implement the function or which can be adapted or modified to implement the function. As in Appendix B, these listings are grouped by design object.

C.1 User Interface

The operations in this object will have to be implemented with newly-generated code. For the most part, this code will invoke code generated by the RAVE package [36].

C.2 Operating System Interface

All the operations can be implemented using function and/or system calls available in the OS-9 C compiler [33], with the exception of Transfer File. That operation also requires a shell call to the kermit application program.

C.3 File Interface

Create, Read, Write, Display, and List can be implemented using functions implemented in the `file.c` package in [39]. Copy, Delete, and Change Permissions can be implemented using C function calls available from the C compiler [33].

C.4 Real-Time Front End Processor Interface

Set Sample Rate, Set Mixer Frequency, and Load Coefficients can be implemented by modification of the `fe.c` package written by SRL [25]. Start Sampling, Stop Sampling, and Validate Parameters will have to be implemented with newly generated code.

C.5 Signal Processing Hardware Interface

All the operations for this object will have to be implemented using newly generated code.

C.6 IV-3272 Interface

Load Configuration, Load DSP Program, Start Transfer, and Stop Transfer can be implemented using the `dma.c` package written by SRL [25], source code provided by Ironics in the `aset_fsdt.c`, `cset_fsdt.c`, and `set_fsdt.c` packages [22], and newly generated code.

C.7 a66540 Interface

Load Configuration, Start FFT, and Stop FFT can be implemented using code provided by Array Microsystems [24].

C.8 Discrete Digital Signal

Create and Display can be implemented using functions implemented in the `file.c` package in [39].

C.9 Signal Sample

No code is required to implement Create.

C.10 Time-Domain Signal

No code is required to implement Fast Fourier Transform. Since the operations for this object are inherited from the superclass Menu, they will be implemented as modified versions of the code implementing the operations in the superclass.

C.11 Frequency-Domain Signal

No code is required to implement Inverse Fast Fourier Transform. Since the operations for this object are inherited from the superclass Menu, they will be implemented as modified versions of the code implementing the operations in the superclass.

C.12 Window

Create and Remove can be implemented using code generated by the RAVE package [36].

C.13 Menu

Create and Remove can be implemented using code generated by the RAVE package [36].

C.14 Choice Item

Create and Execute Function can be implemented using code generated by the RAVE package [36].

C.15 Horizontal Bar

Since the operations for this object are inherited from the superclass Menu, they will be implemented as modified versions of the code implementing the operations in the superclass.

C.16 Pop Up Box

Since the operations for this object are inherited from the superclass Menu, they will be implemented as modified versions of the code implementing the operations in the superclass.

C.17 Data Input Box

Send Data can be implemented using code generated by the RAVE package [36]. Since the other operations for this object are inherited from the superclass Menu, they will be implemented as modified versions of the code implementing the operations in the superclass.

Appendix D. System Operation

This appendix documents commonly-used system operation procedures. System initialization, logging in, new user creation, and file transfer to other AFIT computer systems are covered.

D.1 System Initialization, User Login, and User Creation

To start the PSK system, apply power to the VT-340 terminal, the NEC 5D display, the outboard SCSI peripheral chassis, and the VMEbus cabinet. The order in which power is applied is not particularly important, but applying power first to the display devices ensures that they are warmed up and ready to display as soon as the system initializes.

To initialize the system, type "os" at the IV-3206> prompt. The next prompt displayed will be the system debugger debug: prompt. Type "g" at this prompt and the OS-9 boot sequence will start.

To log in to the system, press carriage return once the initialization sequence has stopped. This will produce a system login prompt. Type "fe" for the username and "fe" again for the password.

All the known users and their passwords are stored in the clear in the `password` file located in the `SYS` directory. To create a new user, edit the `password` file using the μ MACS editor. The `password` file contains the username, password, initial user process priority, default data and execution directories, and default shell.

D.2 File Transfer

Currently, the only method available to transfer files to and from the PSK system is by use of the **Kermit** file transfer package. Furthermore, the only transfer target routinely available is the stand-alone PC also located on the bench on which the PSK system is located. To transfer a file from the PSK system to the PC, perform the following steps:

1. Start both the PSK system and the PC and log in to PCnet using the appropriate username and password.
2. On the PC, escape from the PCnet menu to the DOS command prompt.
3. On the PC, start **kermit** by typing its filename. Press "c" to connect to the PSK system. If it does not respond, check the communication speed to make sure **kermit** and the PSK system are using the same speed.
4. Log in to the PSK system.
5. Change the current data directory to the directory in which the desired file is located.
6. On the PSK system, type "**kermit -s filename**".
7. Escape back to the PC by typing control-left brace, then type "r" to receive the file.
8. Observe the file transfer, which will stop automatically.
9. Press "c" at the PC-Kermit prompt to return to the PSK system.

Theoretically, file transfers from PCnet to the PSK system can be accomplished by reversing the roles of **send** and **receive** in the above scenario. However, there is apparently a cabling problem which causes the PSK system not to recognize the PC is attempting to send a file. Time did not permit this problem to be rectified.

References

1. Systems Research Laboratories, Inc., 280 Indian Ripple Rd., Dayton, OH 45440, *SRL PSK Matched Filter*, June 1990.
2. Y. Jain, "DSP Design Made Easy," *Machine Design*, vol. 62, pp. 88-94, July 1990.
3. W. Todd, "A General Purpose Mini-computer Based Digital Signal Processing Laboratory," MS thesis, AFIT/GE/EE/82J-14, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, Dec. 1982.
4. J. W. Bengtson, "Development of a Real-Time General-Purpose Digital Signal Processing Laboratory System," MS thesis, AFIT/GCS/EE/83D-3, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, Dec. 1983.
5. R. F. Mills, "Real-Time Digital Signal Processing Implementation for the Modulation Detection and Classification (MODAC) Receiver," MS thesis, AFIT/GE/ENG/87D-46, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, Dec. 1987.
6. W. Andrews, "DSP Boards Reach Performance Highs," *Computer Design*, vol. 31, pp. 69-79, Aug. 1992.
7. S. Tesmer, "VMEbus CPU and DSP Boards Team Up for Embedded Control Applications," *Computer Design*, vol. 31, p. 76, Aug. 1992.
8. J. Robinson, "The Evolution of DSP Development Tools," *Computer Design*, vol. 31, p. 98, May 1992.
9. B. C. Mather, "Embedding DSP," *IEEE Spectrum*, vol. 28, pp. 52-55, Nov. 1991.
10. J. Rasure, D. Argiro, T. Sauer, and C. Williams, "Visual Language and Software Development Environment for Image Processing," *International Journal of Imaging Systems and Technology*, vol. 2, pp. 183-199, 1990.
11. W. Hodgkiss and J. Nickles, "Real-Time Data Management in a UNIX Network Environment," in *Conference Proceedings - OCEANS 1990*, (445 Hoes Ln., Piscataway, New Jersey, 08855), pp. 294-297, IEEE, IEEE Service Center, 1990.
12. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*. Englewood Cliffs, New Jersey: Prentice-Hall, 1991.
13. T. Bihari, P. Gopinath, and K. Schwan, "Object-Oriented Design of Real-Time Software," in *Proceedings of the Tenth Real Time Systems Symposium*, (10662 Los Vaqueros Circle, Los Alamitos, CA 90720), pp. 194-200, IEEE, 445 Hoes Ln., Piscataway, New Jersey, 08855, IEEE Computer Society Press, 1989.
14. B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Reading, Massachusetts: Addison-Wesley, 1987.
15. D. Hix and H. R. Hartson, *Developing User Interfaces: Ensuring Usability Through Product & Process*. New York, NY: Wiley & Sons, Inc., 1993.
16. S. L. Smith and J. N. Mosier, "Guidelines for Designing User Interface Software," Tech. Rep. ESD-TR-86-278, MITRE Corporation, Bedford, Massachusetts, Aug. 1986.

17. T. Biggerstaff and C. Richter, "Reusability Framework, Assessment, and Directions," *IEEE Software*, pp. 41-49, July 1987.
18. E. J. Chikofsky and J. H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, pp. 12-17, Jan. 1990.
19. J. D. Coppola, "EENG 699 Project Report Real Time RAM Buffer Memory," Sept. 1992.
20. Ironics Incorporated, 798 Cascadilla St., Ithaca, New York 14850, *IV-3206 VMEbus Single Board Computer and Multiprocessing Engine*, 1990.
21. Chrislin Industries, Inc., 31312 Via Colinas #108, Westlake Village, CA 91362, *CI-VME40 Ultra High-Speed VMEbus/VSB Dual Ported Dynamic Memory Technical Manual*, Nov. 1990.
22. Ironics, Inc., 798 Cascadilla St., Ithaca, New York 14850, *IV-3272 VMEbus Full Speed Data Transporter User's Manual*, 1988.
23. Vigra, Inc., 4901 Morena Bl. Bldg 502, San Diego, CA 92117, *MMI-250 High Resolution VMEbus Graphics Controller*, Nov. 1990.
24. Array Microsystems, 1420 Quail Lake Loop, Colorado Springs, CO 80906, *a66540 VME Compatible Frequency Domain array Processor User's Guide*.
25. System Research Laboratories, Inc., 280 Indian Ripple Rd., Dayton, OH 45440, *PSK Demodulator User's Manual and Specifications*, 1990.
26. S. Heath, *VMEbus User's Handbook*. 2000 Corporate Bl. NW, Boca Raton, Florida 33431: CRC Press, Inc., 1989.
27. NEC Technologies, Inc., Tokyo, Japan, *NEC MultiSync 5D User's Manual*, 1989.
28. Microware Systems Corporation, 1900 N.W. 114th St., Des Moines, Iowa 50322, *OS-9 Technical Manual*, 1989.
29. Microware Systems Corporation, 1900 N.W. 114th St., Des Moines, Iowa 50322, *Using Professional OS-9*, 1989.
30. Microware Systems Corporation, 1900 N.W. 114th St., Des Moines, Iowa 50322, *Using uMACS*, 1989.
31. Microware Systems Corporation, 1900 N.W. 114th St., Des Moines, Iowa 50322, *OS-9 System State Debugger*, 1987.
32. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Englewood Cliffs, New Jersey: Prentice-Hall, 1978.
33. Microware Systems Corporation, 1900 N.W. 114th St., Des Moines, Iowa 50322, *OS-9 C Compiler User's Manual*, 1989.
34. Microware Systems Corporation, 1900 N.W. 114th Street, Des Moines, Iowa 50322, *RAVE Graphics Support Library: A Programmer's Guide*, 1990.
35. Microware Systems Corporation, 1900 N.W. 114th Street, Des Moines, Iowa 50325, *RAVE Graphics File Manager Technical Manual*, 1990.

36. Microware Systems Corporation, 1900 N.W. 114th Street, Des Moines, Iowa 50325, *Using the RAVE Presentation Editor*, 1990.
37. Microware Systems Corporation, 1900 N.W. 114th Street, Des Moines, Iowa 50325, *RAVE IFF Support Library*, 1990.
38. T. Hartrum, *Class Notes for CSCE 594*. Air Force Institute of Technology (AU), Wright-Patterson AFB, Ohio, Oct. 1992.
39. P. M. Embree and B. Kemble, *C Language Algorithms for Digital Signal Processing*. Englewood Cliffs, New Jersey 07632: Prentice Hall, 1991.
40. B. W. Kernighan and P. J. Plauger, *The Elements of Programming Style*. New York, New York: McGraw-Hill, second ed., 1978.
41. K. Johnson, E. Simmons, and F. Stluka, *Ada Quality and Style: Guidelines for Professional Programmers*. 2214 Rock Hill Road, Herndon, Virginia 22070: Software Productivity Consortium, second ed., 1991.

Vita

Captain Jeffrey C. Miller was born 1 December 1961 in Inglewood, California. He graduated from Granada Hills High School, Granada Hills, California, in June 1979 and attended the University of California, Los Angeles from September 1979 to December 1983. He graduated *magna cum laude* from UCLA with a Bachelor of Science degree in Engineering. Captain Miller enlisted in the Air Force in February 1983 under the College Senior Engineering Program and attended Officer Training School from January to March 1984, receiving a commission as a Second Lieutenant. His first assignment was to Griffiss AFB, New York, to the 485th Engineering Installation Group as a Telecommunications Switching Systems Engineer and later as a Standards and Review Engineer. While at Griffiss, Captain Miller attended off-duty education courses given by Rensselaer Polytechnic Institute and was awarded the Master of Business Administration degree in May 1988. In March 1988, Captain Miller was reassigned to the 1843rd Engineering Installation Group, Wheeler AFB, Hawaii, as the Chief of Engineering Workload Control and later as the Chief System Engineer for the Japan Reconfiguration and Digitization Program. In May 1992, Captain Miller entered the AFIT School of Engineering, Wright-Patterson AFB, Ohio, to pursue a course of study leading to a Master of Science in Electrical Engineering degree.

Permanent address: 11042 Babbitt Ave.
Granada Hills, CA 91344

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 1993		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE DEVELOPING A GRAPHICAL USER INTERFACE TO SUPPORT A REAL-TIME DIGITAL SIGNAL PROCESSING SYSTEM				5. FUNDING NUMBERS
6. AUTHOR(S) Jeffrey C. Miller				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/93D-16
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Mr. James A. Wetz NAIC/DXSI 4115 Hebble Creek Rd., Suite 12 Wright-Patterson AFB, OH 45433-5616				10. SPONSORING / MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Distribution Unlimited				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 words) A graphical software user interface for a VMEbus-based real-time digital signal processing system was designed. User requirements were defined and the Rumbaugh object-oriented analysis and design technique was applied to analyze the requirements and produce an object-oriented design. The software design includes a graphical, mouse- and keyboard-driven user interface, specialized hardware driver modules, and operating system interfaces. An implementation plan was also developed to map the design into the C programming language using existing system code, automatically generated code, and newly written code. Based on the implementation plan, a limited software system prototype was successfully developed and demonstrated. The system can be used to analyze signals previously recorded on disk or sampled in real time. Analyzed signals can be displayed either as a set of discrete samples or as a graph in either the time or frequency domains. The system includes real-time sampling hardware, specialized DSP hardware, general-purpose computing hardware based on the Motorola 68030 microprocessor, mass storage media, and a high-resolution graphical display.				
14. SUBJECT TERMS Software Engineering, Interfaces, Signal Processing, Real Time, Digital Computers, Digital Communications				15. NUMBER OF PAGES 162
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	