Design and Development of SINK, a Software INteractions Knowledge System

by

I. Ajit Naidu

Report submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements of the degree of
Master of Science
in
Computer Science and Applications

APPROVED:

Edward A. Fox

Dr. Edward A. Fox, Chairman

Osman Balci

Dr. Osman Balci

John Long

Mr. John O. Long

November, 1992
Blacksburg, Virginia

Design and Development of SINK, a Software INteractions Knowledge System

by

I. Ajit Naidu

Chairman, Dr. Edward A. Fox, Computer Science

(ABSTRACT)

The objective of this project has been to develop the SINK (Software INteractions Knowledge) system based on the client-server paradigm. The SINK knowledge server allows information about interactions between switching systems' software features to be retrieved by users on the Bell-Northern Research (BNR) network. The clients of the SINK system have a X Window System graphical user interface written to conform to the interface design principles laid down in the OSF/Motif style guide. A number of important issues relating to the design of client-server applications and graphical user interfaces are dealt with in this project.

The SINK (Software INteractions Knowledge) system will serve as a central repository of corporate knowledge at Bell-Northern Research about its switching software assets and the interactions between those assets. The SINK system will help increase the productivity for designers and improve understanding about feature interactions as it provides a comprehensive source collected previously from many sources for a wide spectrum of information about existing switching software assets and their interactions.

ACKNOWLEDGMENTS

I would like to thank Dr. Edward A. Fox for his patience, understanding, encouragement and guidance during the course of this work. I sincerely thank Dr. Osman Balci for agreeing to serve on my committee. I would also like to thank John O. Long for agreeing to serve on my committee and for suggesting this work as a feasible project and helping me with a number of aspects of the SINK system. I would like to thank my colleagues Adam Resnick and Eric Schutz at Bell-Northern Research for their assistance during the course of this project.

I would like to thank my wife Dawn, my parents and sisters for their enormous understanding and encouragement.

Table of Contents

List of Illustrations

List of Tables

CHAPTER ONE

INTRODUCTION

1.1 Motivation and Overview

The present period, often called the Information Age, has more information generated about more topics than was ever done before. In this complex world, relevant information is often needed to carry out the tasks at hand and to make intelligent decisions [SALT 83]. The problem of storing and accessing information has taken on an even greater significance on account of the rapid growth of human knowledge which has resulted in a veritable information explosion [THIM 90]. When information is collected and stored, it is difficult to find the data actually needed at a given time, and to distinguish relevant from extraneous data [SALT 83]. For this reason, electronic search aids are widely used to process, store, and retrieve information items on demand.

As the amount of information increases the demands on information storage and retrieval systems increase. At **B**ell-**N**orthern **R**esearch (BNR) there exists an enormous amount of information about their large telecommunication software system. When the telecommunication software system of a telephony switch is initially designed, its components are understood by a small group of designers. Those designers have a deep understanding of the domain and the design rationale. However, as this software system is maintained and extended over time to create new functionality or to make patches, its structure becomes increasingly complex, and the components of the software and the interactions of those components become increasingly obscure to both the initial designer and the new designers who are assigned to work on the system.

There is no easy way of referencing information or of knowing exactly where to go to get all the desired information, and obtaining this information can be a difficult and time-consuming task. Currently, large corporations are increasingly relying on strategic information systems to keep track of corporate software assets. One specific example is the

Large Software System Information Environment (LaSSIE) system being built by AT&T Bell Labs [DEVA 91]. This knowledge-based information system helps users explore the software assets in the AT&T Definity 75/85, a scalable PBX product with a flexible and powerful feature set.

It was with these issues in mind that the **S**oftware **IN**teractions **K**nowledge (SINK) system was designed, a central repository of corporate knowledge at BNR about its telephony software. The aim of this project was to design and implement an OSF/Motif based graphical user interface and provide networked access to the SINK system. The user interface and networking modules have been written to facilitate the task of retrieving information easily and effectively. A formal usability study of the user interface has not been performed, but it has been designed and implemented with continuous interaction with the users.

1.2 The SINK system

The **S**oftware **IN**teractions **K**nowledge (SINK) [LONG 91] system will serve as a central repository of corporate knowledge at BNR about its switching software assets and the interactions between those assets. *Switching software assets* refer to the "components (packages, features, and modules) that make up the software distributed for Central Office switches" [LONG 91]. An *activity* is a software development activity that corresponds to a specific activity ID (ACTID). A *feature* is a software telephony capability or enhancement to an existing activity and is composed of two or more individual software activities. A *package* is a set of activities bundled together for marketing purposes. A *product* is made of two or more packages. Figure 1.1 depicts these assets. Information about these assets is spread about in a variety of archives, databases and various marketing publications. See Appendix A for definitions of important terminology.

The SINK system consists of three main components:

- Knowledge server - accepts queries from users and then sends the results based upon the response received from the knowledge base,
- Graphical user interface - provides a menu-driven OSF/Motif based front-end for the user to interact with SINK,

- Automated knowledge-acquisition system - collects on-line data, creates knowledge base objects, and infers relationships.

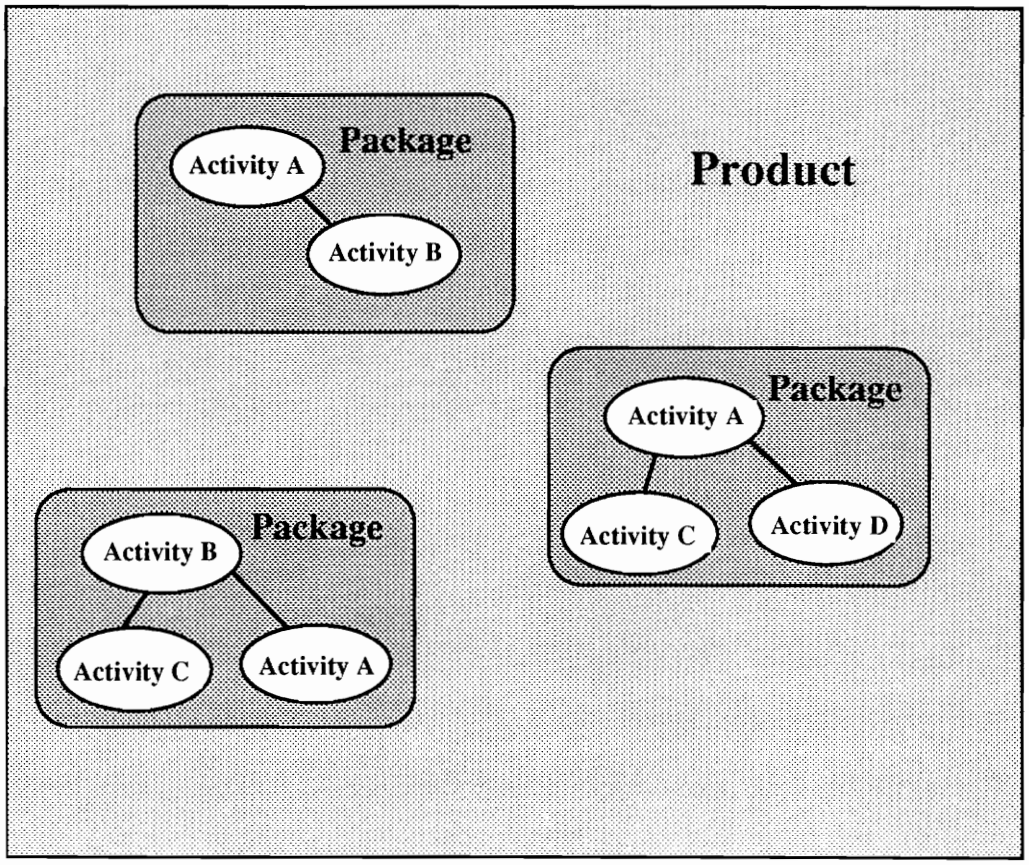

FIGURE 1.1 Switching software assets

After SINK is started, the *main* window is displayed in which the user may browse:

- activities (including ACTID, activity title, package ID, product ID (PRODID), and Batch Change Supplement (BCS) / release),
- features (including feature name and the number of activities in the feature),
- packages (including package ID, package title, BCS/release, PRODID, and the number of activities in the package),
- releases (including BCS/release, the number of packages in the release, and the number of activities in the release),

- products (including product ID, product title, the number of packages in the product, and the number of sub-products contained in the product), and
- undocumented interactions.

After browsing one of these lists, the user may select one item from that list and receive more detailed information about that item. The user may also print this information or save it to a file at anytime. Additionally, the user may obtain context-sensitive help from most windows. The six browsers accessible from the *main window* are described in the following sections.

1.2.1 Browsing Activities

When presented with a list of activities, the user may select an activity and display detailed information about that activity. An activity is a software development activity that corresponds to a specific ACTID. The information that the user may browse will include:

- the summary section of the Functional Description (FN),
- a list of related documents (that is, documents that are mentioned in the FN of any activity that interacts with this activity),
- the restrictions/limitations section of the FN,
- the interactions section of the FN,
- the references section of the FN,
- a list of interacting activities and features,
- features to which the activity belongs,
- the product to which the activity belongs,
- the package to which the activity belongs, and
- the BCS in which the activity was first released; the activity must be present at RTM in that BCS.

1.2.2 Browsing Features

When presented with a list of features, the user may select a feature and display detailed information about that feature. A feature is a telephony capability or enhancement that is composed of a number of individual software activities. The information that the user may browse will include:

- a list of activities which belong to this feature,
- a list of interacting activities, and

- a list of related documents.

1.2.3 Browsing Packages

When presented with a list of packages, the user may select a package and display detailed information about that package. This information will include:

- a list of activities that belong to that package,
- the first BCS in which that package was released, and
- a list of related packages.

1.2.4 Browsing Releases

When presented with a list of BCS releases, the user may select a release and display detailed information about that release. This information will include:

- the list of packages in the BCS release,
- the Program Initiation (PI) date of the BCS release, and
- the Ready To Manufacture (RTM) date of the BCS release.

1.2.5 Browsing Products

When presented with a list of products, the user may select a product and display detailed information about that product. This information will include:

- a list of packages which belong to this product, and
- a list of subproducts that belong to this product.

1.2.6 Browsing Undocumented Interactions

Not all feature interactions are documented. Some are discovered by designers and testers after FN's have been written. The user can enter the undocumented interaction(s) if any in this browser. This list of undocumented interactions include:

- A title of the undocumented interaction. The title may be any descriptive title given by the user, and
- A category. The category is any category string given by the user. The category field is primarily for sorting purposes.

The user may enter a new interaction or edit and display a previously-entered undocumented interaction. All interactions entered by the user are only available to that user. These undocumented interactions can then be sent to the SINK developers as a ascii text

file via email. These undocumented interactions will then be considered by SINK administrators for inclusion in subsequent updates of the SINK knowledge base.

The SINK system should help increase the productivity for designers and improve understanding about feature interactions as it will provide a comprehensive source for a wide spectrum of information about existing switching software components and their interactions collected previously from many sources. It should reduce the response time for support personnel, so they can easily identify the related and interacting features, resulting in improved overall software development/support.

1.3 Contribution to the SINK System

The SINK system is accessible to users on the network because it has been designed and implemented based on the client-server paradigm. The author contributed in the design and development of the user interface and networking modules of the client-server based SINK system. The author's contribution to the SINK system are shown in Figure 1.2. A knowledge server has been developed that can handle multiple simultaneous requests by users on the network. Also the SINK client has been developed that has a graphical user interface based on the X Window System developed to conform to the OSF/Motif style guide[OSF 90a]. The graphical user interface has been designed with importance given to not only aesthetic appeal, but also to facilitate the task of information retrieval, to make it easy and effective.

1.3.1 The SINK Knowledge Server

The SINK package consists of a number of independent programs that perform the various tasks required of an information retrieval package such as indexing of documents, insertion of documents in a collection, and the retrieval of information from these collections based on queries submitted by users [BUCK 85]. On account of the need of the knowledge server to be capable of handling multiple simultaneous requests from users on the network, a stateless concurrent server has been implemented. The knowledge server will run on a Sun SPARCstation which is accessed by clients on the network.

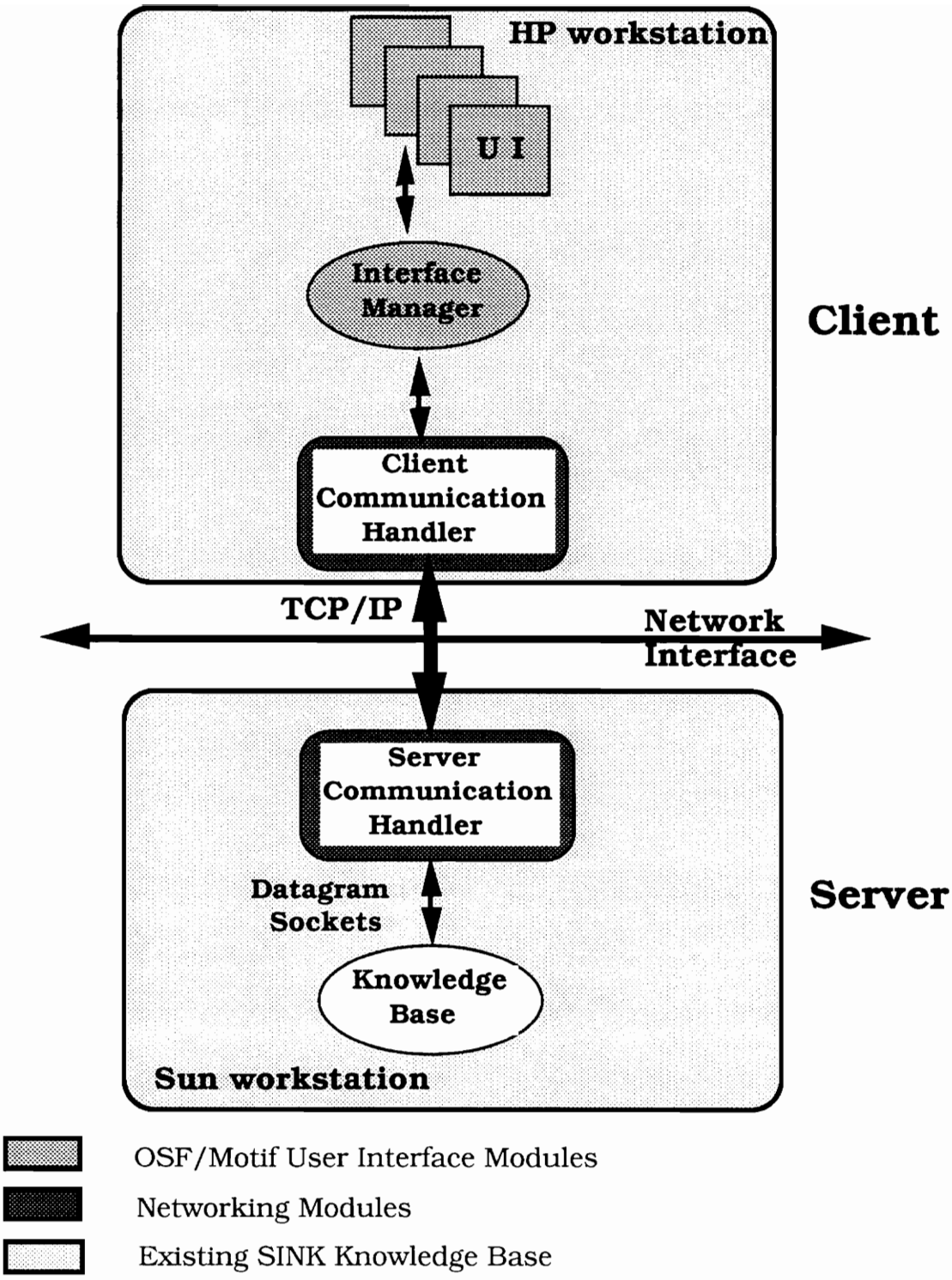

FIGURE 1.2 Contributions to the SINK Architecture

1.3.2 The SINK client user interface

Numerous advances have taken place in the area of user interface design and development during the last few years, most notably the X Window System and the many user interface toolkits [NYEA 89b]. The SINK client (user front-end) has been implemented using the X Window System and the OSF/Motif toolkit.

1.4 Overview of the Report

This first chapter of the report states the objectives of this project and introduces the reader to the different areas in which work has been done. In the second chapter, the literature that was surveyed by the author is briefly discussed. The literature is primarily in the areas of information retrieval, distributed systems (especially the client-server model), human-computer interaction and the X Window system.

In chapter 3, the SINK system is discussed. The major components of the SINK system are explained and how they fit into the client-server paradigm is considered. Chapter 3 also discusses some general issues regarding the design of servers and presents the details of the SINK knowledge server. In Chapter 4, a detailed description of the client and server processes is given. Chapter 5 discusses the OSF/Motif front-end of the clients. In Chapter 6, the client-server communication data format and the methods of the knowledge base objects is described. Chapter 7 summarizes the work done in this project and the conclusions that were arrived at by the author.

CHAPTER TWO

LITERATURE SURVEY

2.1 Information retrieval

To a certain extent, almost all interactions with computers involve a kind of information retrieval - whether it be reading a file, obtaining a list of files, printing a paper copy of a figure, or browsing a network of news groups. A more restricted definition will apply to an area of computing that is devoted to the study of the representation, storage and access of information. However the type of information that is stored is virtually unrestricted.

Information retrieval systems "process files from records and requests for information, and identify and retrieve from the files certain records in response to the information requests" [SALT 83]. A central problem in information retrieval is that of matching the language of the data provider and the user of the data [DUMA 88]. The solution to this problem of information retrieval requires an analysis and classification of the information. The information content can thus be determined and items of information that could meet the information needs of the user are identified.

Many of the initial designs of computerized retrieval systems were based on the practices developed for indexing and retrieval in the paper-ink world of card catalogs where limitations on physical space and people's ability to combine information required that the number of terms and cross references be limited [CLEV 84]. Two salient products of these constraints are the use of restricted indexing vocabularies, and Boolean comparison.

In the earliest retrieval systems, keywords chosen from a restricted set were assigned to each document by professional indexers. Retrieval was performed by trained intermediaries using the same restricted vocabulary. In tests which have compared searching on controlled language index terms with searches on all terms in the full text of the document abstracts, the results have favored the full text methods [CLEV 66][CLEV

84][SALT 73]. An alternative to restricted vocabularies is to allow retrieval using all words or the full-content of the information object. Any word in an object can be used by a searcher to retrieve it. The SMART system by Salton and his colleagues represents an early and systematic effort to understand the benefits and problems of full-content, automatic indexing systems [SALT 83]. Advances in parallel computing have made full-text searching a viable option even for very large databases [STAN 86].

Belkin and Croft [BELK 87] note that research in information retrieval has been moving along three definite directions. The first is that partial match techniques are being used more in operational environments and efforts are being made to relate partial matching to exact matching. Secondly, a mix of techniques seems to present the best solution and there are no techniques that are adequate in all situations. Thirdly, the representations of user's problems are becoming more complex. Although retrieval techniques are different from the representation scheme, those techniques that can be used are dependent upon the representation used. Work in this direction indicates an increased understanding of the importance of the request model compared to the Information Retrieval problem itself.

2.2 Client-Server Model

Developing distributed applications is different from developing non-distributed applications. The fact that the application does not reside in a single process's address space introduces new issues that need to be considered and new tasks that need to be performed. Some of these issues and tasks are:

- which procedures will be local and which will be remote?
- how will the application locate a service on the network?
- what action will the client take when it cannot communicate with the server?
- how will the application handle network communication errors?
- what level of security is needed?
- how will the application be tested and debugged?

The most commonly used paradigm in constructing distributed applications is the client-server model. In this model, the server offers services to the network which the client can access. An application can be a client and a server. Another way to understand the model

is that servers provide resources, while clients consume them. The terms client and server do not necessarily denote computers; they could be thought of as client process(es) and server process(es)[CORBIN 90]. The client-server model of X Windows is illustrated in Figure 2.1

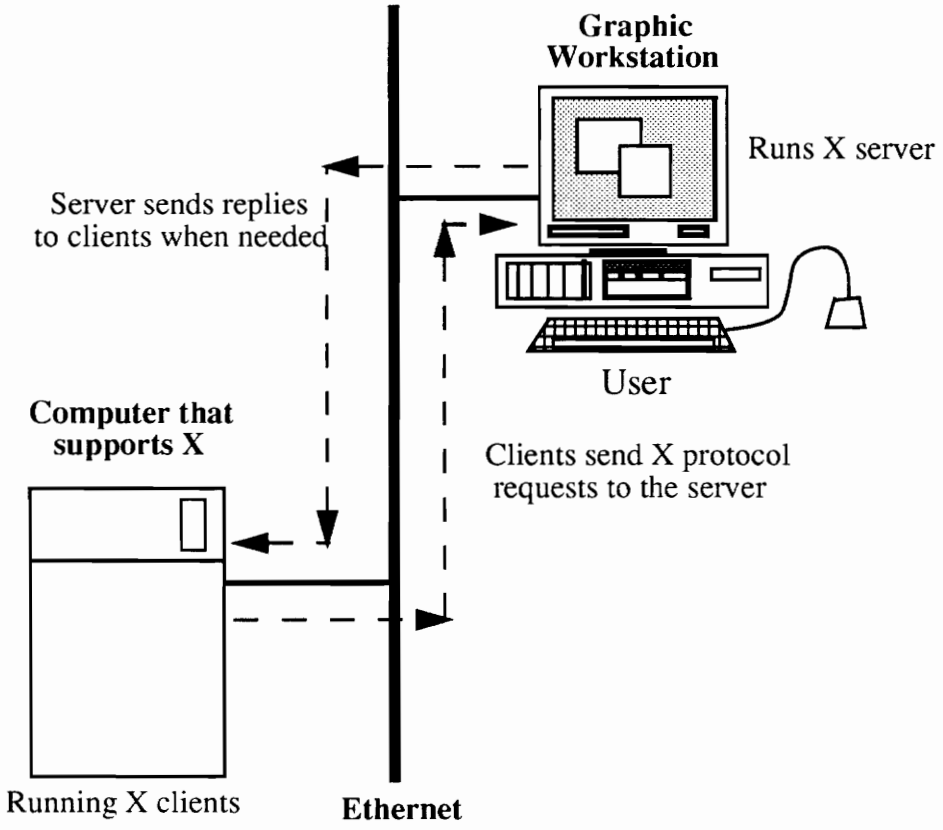

FIGURE 2.1 Client-Server Model

The client and the server require a well-known set of conventions before service may be rendered. This set of conventions comprises a protocol that must be implemented at both ends of the connection. The protocol may be symmetric or asymmetric. In a symmetric protocol, either side may play the master or the slave roles. An example of a symmetric protocol is the TELNET protocol used in the Internet for remote terminal emulation. In an asymmetric protocol, one side is immutably recognized as the master, with the other as the slave. An example of an asymmetric protocol is the File Transfer Protocol, FTP. Whether the specific protocol used in obtaining a service is symmetric or asymmetric, there is a client process and a server process at every point in time[STEV 90].

A server process normally listens at a well-known address for requests. That is, the server process remains dormant until a connection is requested by a client's connection to the server's address. At such a time the server process accepts the connection and performs whatever appropriate actions the clients request of it.

There are two types of servers, *stateful* and *stateless*. A stateful server (that is, one having a state) maintains client information from one request to the next. A stateless server does not need to maintain any state, or information, about any of its clients in order to function correctly. In the event of a failure, stateless servers have a distinct advantage over stateful servers. With a stateless servers the client needs to only retry a request until the server responds; it does not need to know that the server has crashed or that the network temporarily went down [CORBIN 90].

In contrast with file servers and database servers--which are usually processes executing in remote machines--the X display server is a process executing in your workstation while serving clients that may be running on the same or on a remote machine.

A communication protocol is needed so that the request sent by the client is understood by the server and the response spooled back by the server is understood by the client. A protocol is an agreement between the server and the client about how they will exchange information and how that information will be interpreted. The basic building block for communication is the *socket* . A socket is an endpoint of communication to which a name may be bound. Each socket in use has a type and one or more associated processes. Sockets exist within communication domains. Domains are abstractions which imply both an addressing structure (address family) and a set of protocols which implement various socket types within the domain (protocol family). Communication domains are introduced to bundle common properties of processes communicating through sockets[STEV 90].

SINK will use stream sockets for communication between the client and server. A stream socket provides for the bi-directional, reliable, sequenced, and unduplicated flow of data without record boundaries. The user requests are sent to the knowledge server via stream sockets.

2.3 Human Computer Interaction

A new paradigm of computing use seems to be emerging in which computational aid will be applied to the storage, selection, and use of most sorts of information. Although databases and information systems have been around for some time, the systems developed have relied largely on the power of search and indexing techniques. Advances in computer technology have created new possibilities for information retrieval systems in which user interfaces could play a more central role [STUA 91].

In recent years the importance of user interfaces has become widely recognized, and the user interface is a topic which gets increasing attention already at the stage of design of the system [JUNG 90]. *Usability* has become one of the requirements of most software systems. The invention of the concept of windows at Xerox Corporation opened a whole new world of human-computer interaction mechanisms to improve usability. The idea of dividing the display area or screen in smaller viewports or virtual terminals was first exploited by Apple Computer Incorporated.

In a study, Holocomb and Tharp compared three different types of user interfaces namely, windows, menu-driven and command line interfaces [HOLC 85]. These user interface styles were compared based on how well they conformed to principles such as feedback, minimizing demands on human memory, sustaining user orientation and multitasking. They came to a conclusion that the windowing system was far superior to the other interface mechanisms. However they noted that a proliferation of windows tended to confuse the user and there was additional concern about wasted screen real-estate [HOLC 85].

The last decade of research and practice in user interface design has provided some good models for designing user interfaces. Getting input from the users early and continually throughout the design process, using rapid prototyping and iterative design techniques, and conducting formal usability testing are proven methods for assuring good user interfaces [MULL 91]. In the real world, however, it is often difficult to put these principles into practice. Gould and Lewis' [GOUL 85] four principles of designing for usability capture the requirements to create good interfaces.

- early and continual focus on the user tasks,

- interactive design in which potential users of the system participate in the design,
- empirical measurement of usage (i.e., usability testing), and
- iterative design.

These principles are the "motherhood and apple pie" of good interface design [MULL 91]. During recent years usability (in the form of effective access to computer-based function) has become an important issue in development -- in addition to the familiar issues of function, cost, and schedule [HCI 90]. By usability the design of the actions of the various user interface components is referred to so that:

- the actions of the user interface components are truly useful,
- the learning time is acceptable to the intended users,
- the performance of the users is supported, and
- the users choose to continue use of the function.

2.4 X Window System and Toolkits

The SINK system has a X Window System based graphical user interface as the client front-end. The front-end is implemented using the OSF/Motif toolkit. The OSF/Motif toolkit is designed on top of the X Window System. A brief description of both the X Window System and the OSF/Motif toolkit is given below.

The X Window System, called X for short, is a network-based graphics window system that was developed at MIT in 1984. Several versions of X have been developed, the most recent of which is X Version 11 (X11), first released in 1987[REIL 89]. The X Window System architecture is divided into two distinct parts: display servers that provide display capabilities and keep track of the user input, and clients, i.e., application programs that perform specific tasks. The general structure of an X application is illustrated in Figure 2.2.

The X11 Window System defines a distributed, asynchronous protocol [SCHE 87] by which the display servers and applications communicate. A display server can support multiple applications, and applications can use multiple servers. An application and a server can run on the same or on different machines. The X protocol is a communication

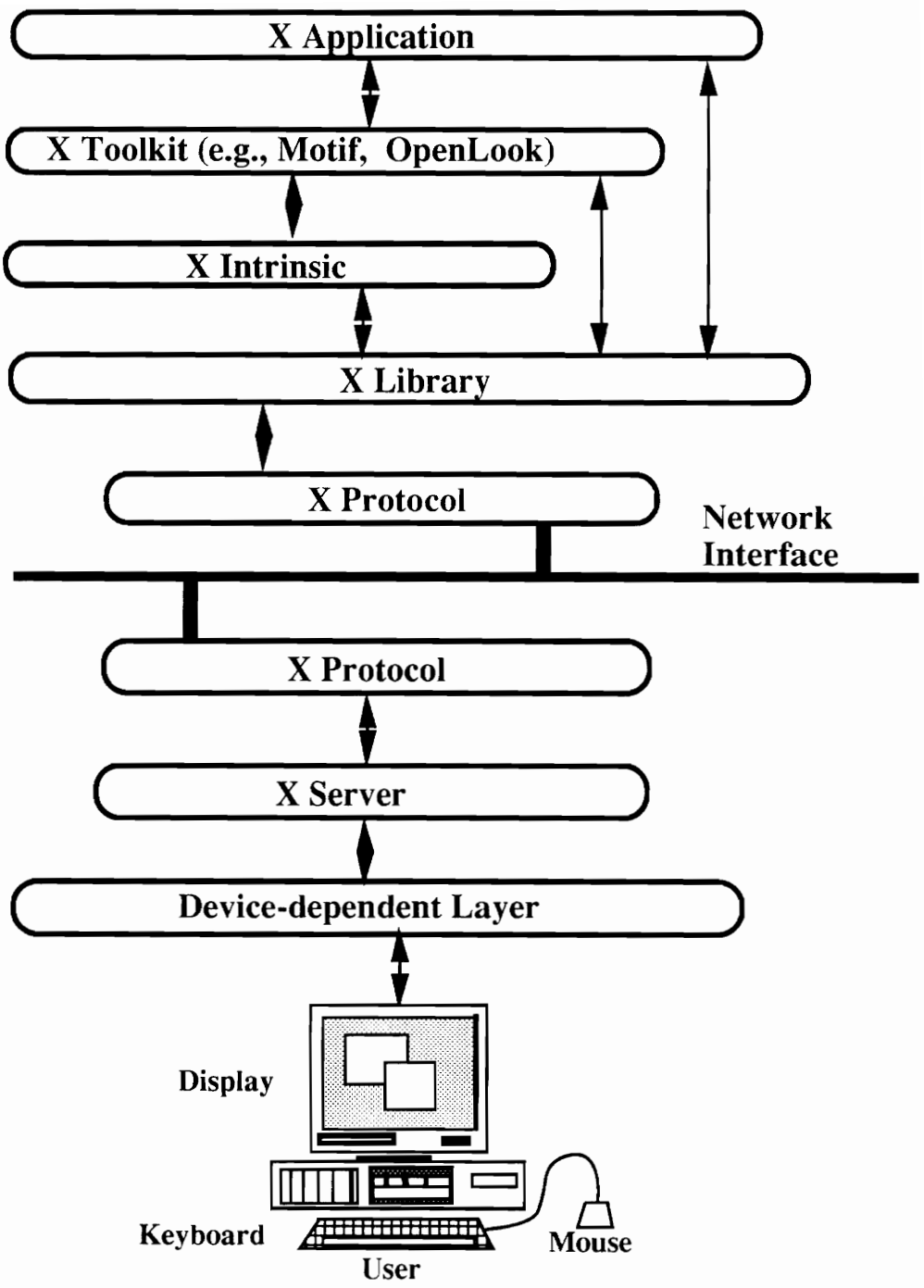

FIGURE 2.2 General Structure of an X application [BARK 91]

protocol that defines how window information can be transmitted and received across a network connection [NYE 88a]. The X protocol assumes a server and an application are connected by a fast communication link like shared memory, Ethernet, or even a leased line [McCOR 88]. The network protocols over which X is currently supported are TCP/IP, DECNet, etc. To minimize the effects of network latency, the X protocol is asynchronous: neither the server nor the application waits for acknowledgments. The binding of the X protocol and the C language is called the X library or the Xlib [NYE 88a]. The Xlib provides a procedural interface to the protocol.

The X library provides a powerful low-level interface, but this flexibility has a cost: even simple programs are hard to write. With a view toward promoting software reusability, modularity, data abstraction and object oriented programming paradigms, the X Consortium (the group that now controls the development of the X Window product) developed the X Intrinsic [ASEN 90]. The X Intrinsic, referred to as Xt, defines the architecture but not the look and feel of the user interface toolkits [ASEN 90]. These toolkits are meant to provide the user with a "simplified approach to graphics programming" [ASEN 90][NYE 89a].

The Xt Intrinsic also known as X Toolkit [McCOR 87] is an object-oriented C language construction kit built on top of the X library. It defines the "how" in creating interface components and provides the library of functions for working with the components. The primary goal of the X toolkit [McCOR 87] is to reduce the effort needed to write an X application. A secondary goal is to allow customization of applications by setting widget resources. Finally, when an application synchronously queries the server, hundreds of these queries quickly arriving can add up to cause delays of several seconds in part due to the latency inherent in a round trip. The toolkit extensively caches data on the application side to minimize this time[McCOR 88].

The Intrinsic layer of the toolkit is a mostly policy-free foundation upon which widgets and applications are built. An application uses the intrinsics to tie the widgets together into a user interface and to bind this user interface to functions implemented by the application. Applications typically interact with X only through the toolkit [McCor 88].

There are a number of user interface toolkits that are available today that conform to the X Intrinsics architecture [NYE 89a]. These toolkits provide a set of user interface objects called widgets. A widget is implemented using calls to the intrinsics and the X library [McCOR 88]. The widget set and its associated interface and convenience functions constitute the toolkit. The OSF/Motif toolkit is one such package [OSF 90d]. It has a large widget set with a full complement of associated convenience routines. The OSF/Motif widgets have a 3-D look and feel due to the effective use of the concepts of shadow and light. It also comes with a powerful window manager, the Motif Window Manager, also called "mwm".

The X Window System and OSF/Motif provide a consistent look and feel to interfaces that are independent of hardware platforms, operating systems and graphics devices. Besides, these systems are relatively easy to use and provide modularity and a high degree of re-usability. The following chapters discuss the SINK system in detail.

CHAPTER THREE

THE SINK SYSTEM

3.1 Overall System Architecture

One of the objectives of this project has been to set-up an online retrieval system acting as a retrieval server so that multiple users can access this server via a graphical user interface from remote sites connected through a network to the server machine.

In order to accomplish these objectives, a graphical user interface for the SINK system was designed and implemented as described in Section 3.2. In addition a knowledge server was also developed so that users could submit queries from a remote workstation. The server would receive the queries and send the responses back to the user.

SINK consists of the Client that has a Motif graphical user interface, a Knowledge Server and a Knowledge Acquisition subsystem. The Client-Server model was selected because of the costs associated with deploying third-party software, namely IBUKI Common Lisp Object System (CLOS) used by the Knowledge Base. Without a Client-Server architecture, the third-party software would have to be distributed throughout BNR, thus incurring considerable cost. Since SINK provides the user with read-only information, no status or session information needs to be maintained by the server, so a *stateless* server has been implemented. The Knowledge Server runs on a Sun SPARCstation and handles requests for information from all clients. The clients run on HP workstations. The following sections describe the architecture of the Server and Clients. The overall Client/Server architecture is depicted in Figure 3.1.

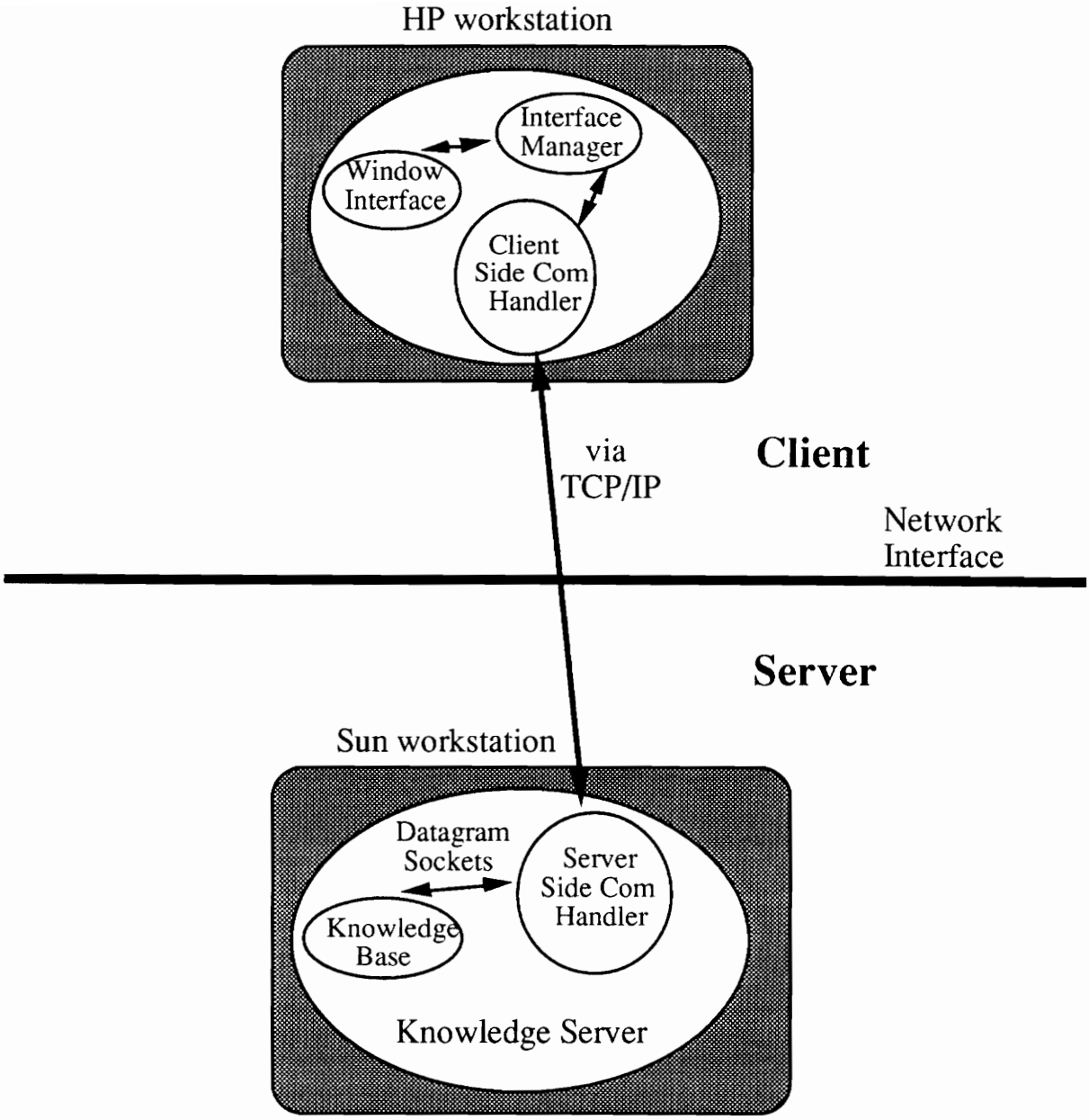

FIGURE 3.1 Overall SINK Architecture

3.2 The SINK Knowledge Server

The Knowledge Server consists of 2 main components: the *Server Side Communications Handler*, and the *Knowledge Base*. The Server Side Communications Handler receives requests from Clients, interprets those requests, and sends the translated request to the Knowledge Base. The Server Side Communications Handler is implemented as a set of C routines called by a Common Lisp Object System (CLOS) program [IBUK 91]. The Knowledge Base processes the Client requests, which are typically requests for information about an activity, and returns the appropriate reply. The Communication Handler packages the reply into a format understood by the Client, and sends it.

The SINK Knowledge Server is based on Berkeley sockets. "Berkeley sockets" refers to an application programming interface to communication protocols that enables processes in the UNIX operating system to communicate with each other [STEV 90]. A server that is based on the Remote Procedure Call (RPC) with the same functionality would be simpler to implement and more flexible. This is because in an RPC a process on the local system invokes a process on a remote system and the eXternal Data Representation (XDR) facility can be used which allows a great deal of flexibility in transferring various types of data across the network. The client side uses an encoder to encode a data type that needs to be transferred and on the server side a decoder routine decodes the data received. The RPC mechanism was not chosen to implement the Knowledge Server as it is not completely portable. The three commonly used RPC implementations are the following [STEV 90]:

- Sun Microsystems' Open Network Computing (ONC). Two parts of ONC are RPC, the Remote procedure Call specification, and XDR, the eXternal Data Representation standard. Sun RPC typically uses either UDP or TCP as the transport protocol.
- Xerox Courier. This is an early implementation of RPC. It includes an RPC model, built on XNS SPP as the transport protocol, and a data representation also.
- Apollo's Network Computing Architecture (NCA). NCA/RPC specifies a remote procedure call protocol and NDR, Network Data Representation, defines a data representation standard.

An application implemented using Sun RPC would require changes in order to run on a system that comes with the Apollo's NCA. This is not the case with sockets [STEV 90]. Birrel and Nelson [BIRR 84] present some performance numbers for RPC on an Ethernet LAN. Their results were obtained for various numbers of arguments and results. For a procedure that is called with two 16-bit arguments and returns two 16-bit results, the difference between a local and remote procedure call was a factor of 100 (two orders of magnitude). For a procedure with an argument of 40 16-bit words and a result of 40 16-bit words, the remote call increased the time by a factor of 33.

With the above mentioned arguments in mind the SINK Knowledge Server was implemented using 4.3BSD sockets with TCP as the transport protocol. The Knowledge Server has been tested on a Sun SPARCstation 2 running SunOS 4.1 which is a derivative of BSD 4.3. The UNIX workstations were the SINK client and server reside are linked by an ethernet LAN connection. The server should run on any UNIX based system that supports sockets. Since most UNIX flavors today support sockets, portability is assured.

When the Knowledge Base is started, it is populated from data in a large number of Object Files. The Object Files contain the LISP constructs that represent knowledge base objects when they are loaded into the Knowledge Base. The Knowledge Base contains both the objects as well as functions for retrieving that information. The Knowledge Base is implemented in IBUKI CLOS [IBUK 91]. The other object oriented environment considered for the Knowledge Base was C++, but IBUKI CLOS was chosen as it has a distinct advantage over C++ in its ability to handle long text strings. The response from the Knowledge Base to a user query is in the form of a single character string which could be as big 4 KB at times. IBUKI CLOS takes care of this problem thus avoiding having to handle it as a special case in the application program as would be the case in C++.

One specific problem faced during the early stages of the implementation of the SINK Knowledge Server needs to be discussed. Even though the server communication handler has been implemented as a set of C routines, it is bundled with the Knowledge Base which is implemented in CLOS. The C routines are called through the CLOS program as needed. Hence for all practical purposes the Knowledge Server is considered to be implemented in CLOS. Since the Knowledge server is designed to be concurrent in nature,

a copy of the complete Knowledge Server is *forked* whenever a connection is requested from a Client. During development when the size of the Knowledge Base was only about 5-6 MB this was not really a issue as the Sun SPARCstation used for development had 32 MB of main memory. While testing for the concurrency of the Knowledge Server a maximum of 2-3 Clients were connected at a time and the performance was not adequate. But as the size of the Knowledge Base grew the performance dropped rapidly. The reason was that when the Knowledge Server ran it loaded the Knowledge Base and the CLOS interpreter environment into the main memory, and every time a request for connection came from a client it would fork a copy of the complete Knowledge Base and the CLOS interpreter environment. This oversight was quickly rectified and the new Knowledge Server actually has a stateless non-concurrent server built into the Knowledge Base and a stateless concurrent server which accepts connections from the Clients.

When a Client requests a connection from Knowledge Server using connection oriented stream sockets, a copy is forked and the Knowledge server is ready to accept another connection. The copy of the Knowledge Server which connected to the Client is now responsible totally for that client. When it receives a query from the client it sends it to the Knowledge Base using datagram sockets and sends the response back to the client using stream sockets. This way at any instance of time there is only one instance of the Knowledge Base being accessed by multiple Clients.

3.3 The SINK Client

The Client process is generally the active element in the client-server model [STEV 90]. While the Server is a passive entity which does nothing until it receives a request from the Client, the Client initiates the connection and awaits the results from the Server. However the Client does not block until it receives the results. The user may continue to send more queries, etc. The Client code is linked into the graphical user interface.

The Client program contains three main components: the *Interface Manager*, the *Window Interface*, and the *Client Side Communications Handler*. The Interface Manger controls the display of windows to the user and passes requests for information to the Client Side Communications Handler. Information that is passed from the Client Side Communications Handler (typically as a result of an earlier request for information by the

user) is received by the Interface Manager and is displayed to the user in a window. Windows are brought up as needed by the Interface Manager. The Client Side Communications Handler takes requests for information from the Interface Manager and sends those requests to the Knowledge Server. Correspondingly, the Handler receives information from the Server, interprets that information, and sends the interpreted information to the Interface Manager. Communications errors are sent to the Client Error Log.

Since the Client Communication Handler is embedded into the OSF/Motif code, some extra attention needs to be given to the processing of information to and from a socket as a regular X Windows or OSF/Motif program runs within an infinite loop waiting for X events and will ignore any data that needs to be read or written to a socket or pipe. One way of solving the problem is to check and see if any data needs to be read or written to the socket or pipe before dispatching every X event. However, this will slow the whole process and is somewhat redundant, as in a typical session the user will make queries at random intervals of time and multiple X events can be generated within that period.

At the X Windows Intrinsic layer a facility is provided called "XtAddInput". This procedure takes the socket descriptor or file descriptor, a read or write input mask and a procedure which will be invoked when data is read or written to the socket or file descriptor. This is used in the SINK Client Communication Handler with a read input mask. Thus whenever there is data to be read from the socket the program interrupts the X Mainloop and processes the data by invoking the procedure which is also an argument of "XtAddInput".

One of the main concerns in a client-server based system is that for the system to function effectively both the client and the server should be up and running. If either one of them dies or the network goes down midway through a transaction, the other side will be unaware until it times out, since the other end is not responding. In the SINK Client and Knowledge Server a socket option, namely "SOL_KEEPALIVE", is used which enables periodic transmissions on a connected socket, when no other data is being exchanged. If the other end does not respond to these messages, the connection is considered broken and an error value is returned. This way the SINK user can be warned in case the network or SINK Knowledge Server goes down.

CHAPTER FOUR

SINK SYSTEM DESIGN

4.1 Detailed Description

A data flow diagram for the overall system is shown in Figure 4.1, "Overall Data Flow Diagram,". The Knowledge Server will be a process running on a Unix server machine. When the Knowledge Server starts, it loads all of its knowledge base objects from a list of object files. A user may start up the SINK Client program from an HP workstation. The user may then do any of the following through the Client:

- make requests of the Knowledge Server. A request is a submitted query for information that resides in the knowledge base. Requests are sent from Clients to the Knowledge Server. When a request is processed by the Knowledge Server, the information is sent back to a Client as a reply,
- print out information in the current window,
- save information in the current window to a file, and
- quit from the system.

When the user makes a request, it is sent via TCP/IP to the Server. The Server then records the request in the Request/Reply Log, processes the request, records request errors (if any) in the Server Error Log, and sends the reply back to the Client via the same TCP/IP connection. The Client then displays the reply on the user's display terminal. Incomplete or syntactically incorrect replies are summarized in the Client Error Log. If the user wants to print out information, that data is output to the printer. If the user wants to save information to a file, that data is stored in the file specified by the user.

After the socket conection is established with the Knowledge Server a shell script invokes the Application Performance Monitoring System (APMS). When a user quits the SINK Client, data is sent to the APMS Server which specifies that the SINK system has been

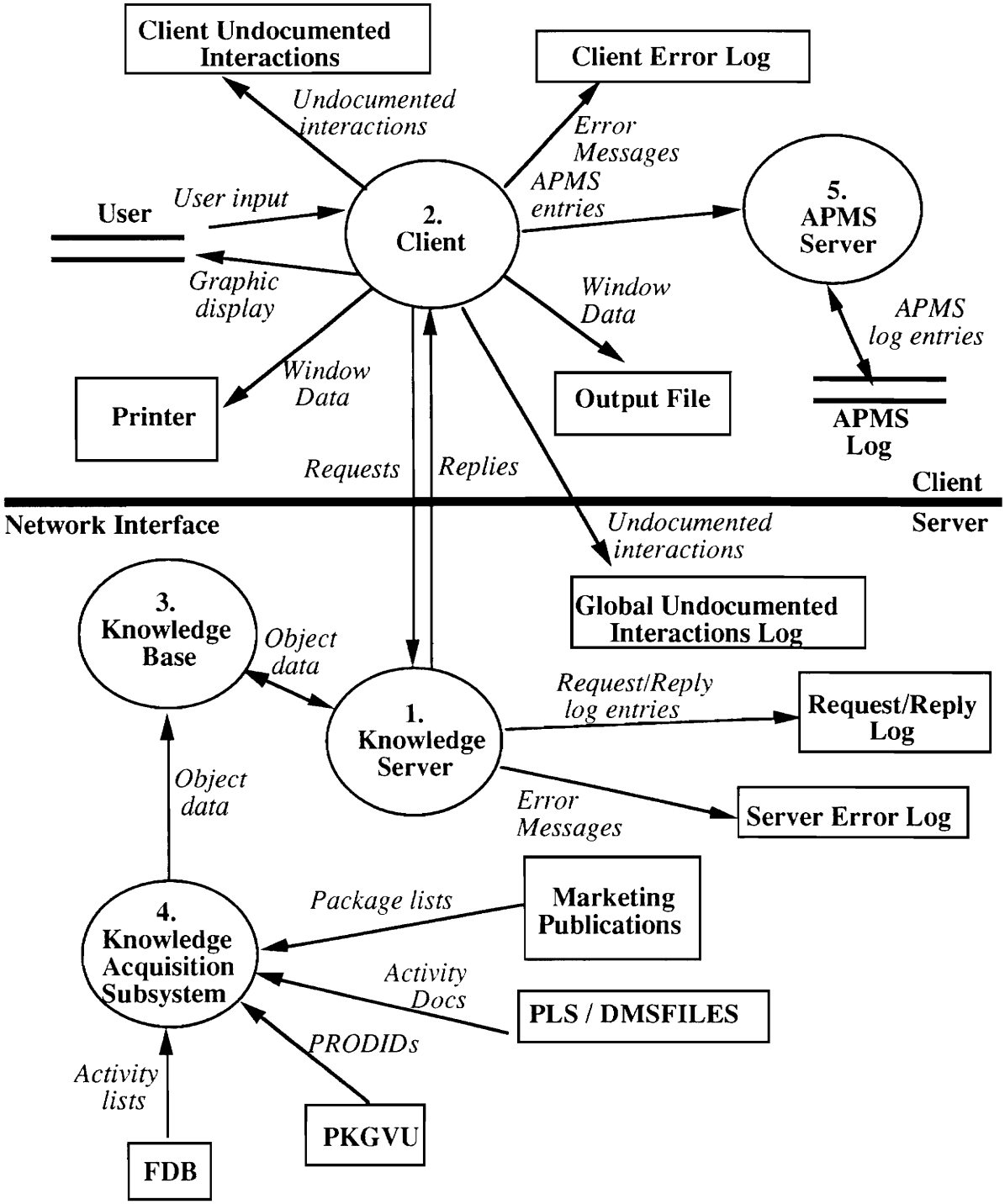

FIGURE 4.1 Overall SINK System Data Flow Diagram

used. The APMS Server then stores that information in the APMS Log. The information in APMS will be used to determine system usage. See Appendix E for details on APMS.

4.2 The Knowledge Server

The Knowledge Server is a continuously running process that listens for requests, receives requests, processes those requests, and returns information. When it receives a request from a Client, the Server "forks" a copy of itself (a duplicate copy of the Server is created) which establishes a connection with the Client. The original Server then returns to the process of listening for requests at its well-known port and the Client communicates with its own Server (the duplicate copy of the Server).

The steps followed after receiving the Client-query to sending back the reply are shown in the data flow diagram of the Knowledge Server shown in Figure 4.2, "Knowledge Server Data Flow Diagram". The following sections (4.2.1 - 4.2.9) describe each of the procedures shown in Figure 4.2.

4.2.1 Receive Request

Once a connection has been established with the Client, the Server receives a buffer consisting of the request from the Client. The Receive Request Procedure reads this buffer from the socket. Each request has the format of a LISP function call. Once the buffer with the request is received in its entirety, it is forwarded to the Log Request procedure.

4.2.2 Log Request

The Log Request Procedure takes a request from the Receive Request Procedure and sends a copy of the request to the Request/Reply Log. The request is sent in the exact form in which it was received. After logging the request, it is given to the Validate Query Procedure. For each request received from a Client, the following data shall be recorded in the Request/Reply Log:

- time and date of inquiry,
- node ID of Client, and
- request in its entirety.

4.2.3 Check Request

The Check Request Procedure receives requests from the Log Request Procedure. Each request contains two components:

- a knowledge base query - a LISP function call which, if executed, is expected to retrieve information from the knowledge base, and
- a window code - an integer value that is used by the Client to determine which type of window to display the Server's reply in.

The Check Request Procedure determines if

- the request contains both a window code and a query,
- the query is a valid LISP function call, and
- the window code is a valid window code.

If any of these three conditions is not true, then the request is said to be an invalid request, the invalid request is output to the Server Error Log, and an error message and the request are both sent to the Package Error Message Procedure. An entry in the Server Error Log will consist of the following information:

- Time and date of error, and
- Error message (which will also contain the invalid request).

If the query is valid, then the query is passed to the Process Query Procedure.

4.2.4 Process Query

The Process Query procedure retrieves information from the SINK knowledge base. This knowledge base is populated when the Knowledge Server is activated by the Initialize Server procedure. The Process Query procedure executes the LISP function call passed from the Check Request Queue procedure. If the query returns information without any errors, then those results are sent to the Package Results Procedure. Otherwise, an error message is output to the Server Error Log and is sent to the Package Error Message Procedure. An entry in the Server Error Log will consist of the following information:

- time and date of error, and
- error message (which will also contain the invalid request).

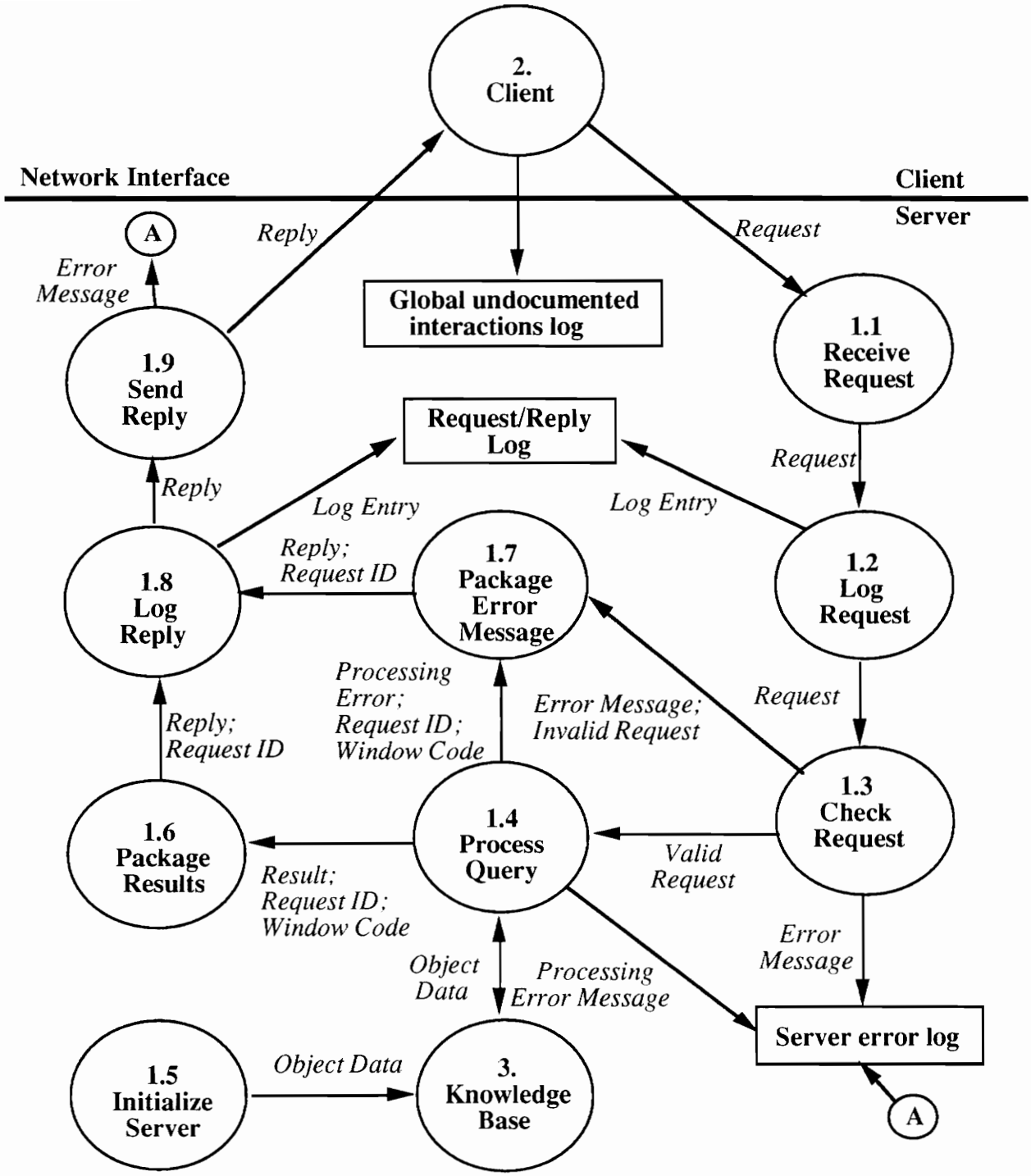

FIGURE 4.2 Knowledge Server Data Flow Diagram

4.2.5 Initialize Server

The SINK knowledge base is populated when the Knowledge Server is started by the Initialize Server Procedure. The Object Files are created by the Knowledge Acquisition Subsystem periodically. These Object Files contain all of the data and information needed to handle all Clients' requests. The information in the knowledge base is used by the Process Query Procedure.

4.2.6 Package Results

The results of the query handled by the Process Query Procedure is sent to the Package Results Procedure. This procedure will add the window code to the query results so that it can be sent as a reply to a Client. Chapter six describes the format of the results of each of the queries processed by the knowledge base.

4.2.7 Package Error Message

If an error message is to be sent to a Client because of an invalid request or a reply error, the Package Error Message Procedure will wrap the error message with the appropriate header information so that it can be sent as a reply to the Client.

4.2.8 Log Reply

The Log Reply Procedure will send a synopsis of each reply (sent back to the Client) to the Request/Reply Log. Because the reply may be lengthy, only a summary of the reply is sent to the log. For each reply sent to a Client, the following data will be recorded in the Request/Reply Log:

- time and date of reply,
- node ID of Client,
- ID of corresponding request, consisting of a client-node/time/date stamp, and
- summary of reply.

4.2.9 Send Reply

This process sends the packaged reply, i.e., the query results and the window code are sent back to the Client who made this request. This packaged reply will be sent to the Client through the socket connection established at the start of the session. This socket connection will exist until the user quits the Client session. The only exceptions are the

network going down or the Client/Server machine going down. In which case suitable error messages will be logged in the Server Error Log.

4.2.10 Communication with Clients

The SINK knowledge server has been designed to be stateless since it provides read-only information to the Client. SINK will use stream sockets for communication between Client and Server. A stream socket provides for the bi-directional, reliable, sequenced, and unduplicated flow of data without record boundaries. The Send Request Procedure (see Section 4.3.5) of the Client process will send a request to the Server via stream sockets. The request will have the following syntax:

Client_Request(magic_number, time_stamp, window_code, query)

In the Client_Request definition shown above, query is the actual modified query which the user makes from the front-end of the Client side. A list of all possible queries is shown in Appendix A. The query needs to be modified because different parts of the Sink Project are written with different programming languages: The Client side is written with the C language, the Server Side Communication Handler within the Server is coded with LISP and C, and the Knowledge Base is written in LISP. The data format conversion required between the Client and Server processes will be handled within the Client, avoiding burdening the Server.

The window_code will be an integer value added by the Interface Manager to identify the type of window to be created by the Interface Manager. These window types and their identifying integer values are pre-defined. Thus, the various windows on the Client side can be created dynamically and the user will be able to perform asynchronous tasks, i.e., the user can send multiple queries even before the reply to the first query returns. The magic_number is the socket connection port number at Client start-up that uniquely identifies that Client. The time_stamp will be the date and time of when that query was created by the user.

When the Server is ready to send information to the Client, it sends that information in the form of a reply. A reply is in the following format:

Server_Reply(magic_number, time_stamp, window_code, result)

A list of all possible reply types is shown in Chapter 6.
The window_code, magic_number, and time_stamp are like their counterparts in the Client_Request described above.

4.2.11 Server Logs

There are two types of SINK Server logging: the logging of normal Server activity in the Request/Reply Log, and the logging of error conditions in the Server Error Log. Information contained in these logs will be used to assist in debugging as well as to develop metrics describing system use.

4.2.11.1 Request/Reply Log

Normal Server activity consists of receiving, processing, and replying to requests from Clients. For each request received from a Client, the following information is recorded in the Request/Reply Log:

- time and date of inquiry
- node ID of Client
- request in its entirety

For each reply made by the Server to a Client, the following will be recorded:

- time and date of reply
- node ID of Client
- ID of corresponding request
- summary of reply

4.2.11.2 Server Error Log

The Client/Server nature of SINK makes it imperative that the Server be able to continue processing without intervention even when errors occur. This imposes the requirement that the Server anticipate these conditions and recover from them whenever possible. For example, it is expected that the Server will verify the existence of a file before it attempts to open and read from it. Should the file be absent, the Server will record this fact in the Server Error Log, reject the inquiry, and prepare itself to process the next incoming request. Another example is a request from a Client about an unknown activity. The

Server would determine the inconsistency, record the error, and return for the next request. It is crucial that these errors are anticipated, detected, and recovered from under normal program flow without being trapped by the LISP condition handlers. Some errors, however, are beyond the control of the Server and cannot be foreseen. These are errors which prevent continued execution under normal flow of control and instead transfer control to the LISP condition handling system. Errors in this category include stream and arithmetic errors. The default behavior is to halt processing, drop into the debugger, and wait for intervention. The Knowledge Server, however, must attempt to continue processing inquiries. Therefore, the condition handlers must be extended to log the specifics of the error and try to restart the Server so that it may continue processing new inquiries.

When an error is detected, an entry will be made in the Server Error Log. An entry in the Server Error Log will consist of the following information:

- time and date of error
- error message

The Server Error Log is an ASCII file.

The amount of disk storage required for Server logs is dependent upon the number of inquiries received from Clients. Each inquiry will result in two log entries; one to record the inquiry, and another to record the reply sent back to the Client. A rough estimate is that each entry will require approximately 100 bytes, or 200 bytes per inquiry. If we assume an average of 15 inquiries per hour for 10 hours a day, then Server log entries will be on the order of 30 K bytes per day, or 150 K bytes per week. A program will be run periodically to purge the Server Logs when they exceed 1 Megabyte of memory.

4.3 Client

The Client functions asynchronously. Once a connection is established with the Server, and the windows are displayed, the Client is ready to accept requests from the user. As replies are received from the Server, information is displayed to the user. The user will not have to wait for a reply before sending another request.

The Server and Client connect via 4.3 BSD sockets. TCP/IP is the specific protocol used to transfer information over the network. Sockets are the tools used by the Client and Server to obtain the information sent and received.

The steps followed after the user starts SINK and sends a query to receiving the result back from the Knowledge Server are shown in the data flow diagram of the Client shown in Figure 4.3, "Client Data Flow Diagram". The following sections describe each of the procedures (4.3.1 - 4.3.8) shown in Figure 4.3.

4.3.1 Start SINK

This procedure is activated when the user starts the SINK interface. This procedure is activated only once during a session. This procedure creates the Main Window for the user to interact with SINK and passes control to the Display Interface procedure. From this point, the user interacts with the Display Interface procedure directly for the rest of the SINK session.

4.3.2 Display Interface

The Display Interface procedure is the main window through which the user interacts with SINK. The other windows are generated to show the information requested by the user. When the user starts the SINK Client, the Main Window will be displayed and a connection with the Server will be attempted. Once the connection is successfully established the user can begin to make queries. These queries are passed to the Package Request procedure.

When a Client session is ending, the Display Interface procedure will log Client information in the Application Performance Monitoring System (APMS). The goal of SINK Client logging is to gain information about system usage. This includes identifying who uses SINK, and how often it is used. Just prior to terminating, the Display Interface procedure will issue a call to the APMS Server with all of the Client information. Daily reports detailing system usage are produced by running APMS software against these logs. For each SINK Client, the following data shall be recorded daily by APMS.

- client's name
- client's department
- Unix login ID

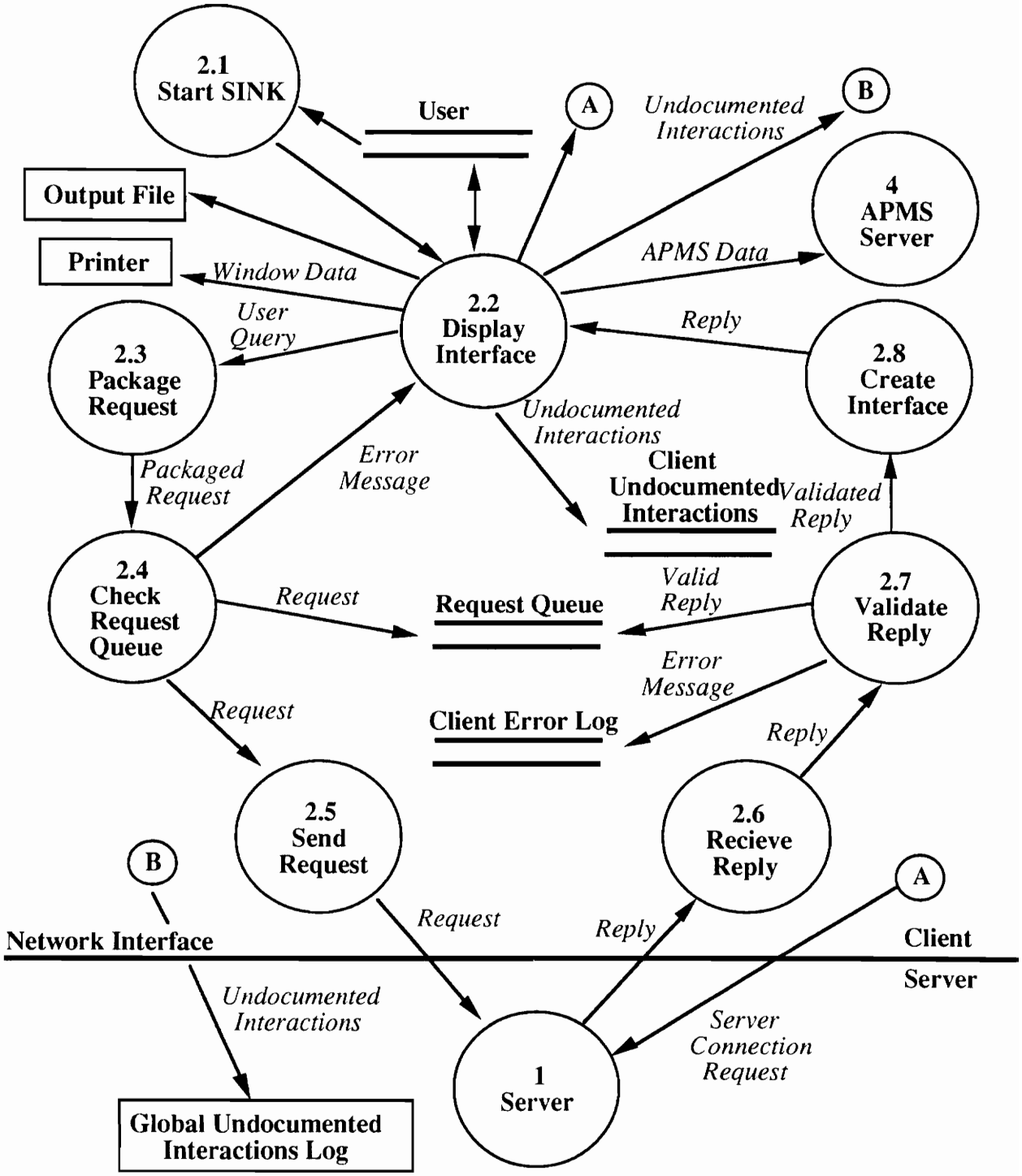

FIGURE 4.3 Client Data Flow Diagram

- workstation node name
- workstation model
- total number of sessions initiated by Client that day
- average duration of sessions

4.3.3 Package Request

This procedure converts the user-query to a data format which the knowledge base will understand by appending it with the following:

- A magic number which is the socket communication port number that is known after a connection is established with the SINK Knowledge server and it uniquely identifies the client.
- A time stamp generated from the system clock.
- A window code which is a pre-defined integer value assigned to each type of window of the user interface. This window code will be used to determine which window the Server reply will be displayed in.

The formatted query with the three additions mentioned above are packaged together. This package is called the request and is sent to the Check Request Queue procedure.

4.3.4 Check Request Queue

This procedure checks the Request Queue for the number of requests pending. If the number of requests pending becomes too high, i.e., exceeds a number which is to be determined, a message is displayed to the user and the user is prevented from making requests [temporarily]. The cursor will turn into an hourglass (during which period the SINK Interface will not accept any input from the user) and the Server will have time to return requests. If the Request Queue has not reached its limit of requests pending, then the request is stored in the Request Queue and is sent to the Send Request Procedure.

4.3.5 Send Request

This procedure sends a request from the Client to the Knowledge Server. The SO_KEEPALIVE socket option [STEV 90] has been used in this Client procedure. This will send a packet of 4 bytes across the socket connection when it is not being used by a SINK Client activated read or write operation. In case the server does not respond then the user will be informed that either the server is down or there is some network problem.

4.3.6 Receive Reply

This procedure receives replies from the Knowledge Server. Replies are passed to the Validate Reply Procedure. This procedure will get activated whenever there is some information available to read in the socket.

4.3.7 Validate Reply

This routine will check to see if the reply has been received completely and will also check the magic number to verify that the Client has received the correct packet. It will then compare the time stamp of the reply with that of the queries in the Request Queue and remove the matching query in the Request Queue. The syntax of the reply will also be checked to determine if the information is suitable to be displayed in a window. It will then modify the reply and remove the magic number and the time stamp. This valid reply is then sent to the Create Interface Procedure. Invalid replies will be sent to the Client Error Log.

4.3.8 Create Interface

This procedure will check the window code and create a window of the required type. The reply received from the Server will be displayed in this window. This window is display by the Display Interface Procedure.

4.4 Knowledge Acquisition Subsystem

The Knowledge Acquisition Subsystem is the process that obtains all the package and activity information needed to build the knowledge base. This process is depicted in Figure 4.4, "Knowledge Acquisition Data Flow Diagram". Listed below are the major steps involved in the Knowledge Acquisition Process:

- A list of packages is manually compiled from the DMS-100 Meridian Digital Centrex marketing publications, such as *VOICE*, *DATA*, *ACD*, and *ISDN*. For each of the packages, the activities are found. The package lists are sent to a workstation where the knowledge server resides.

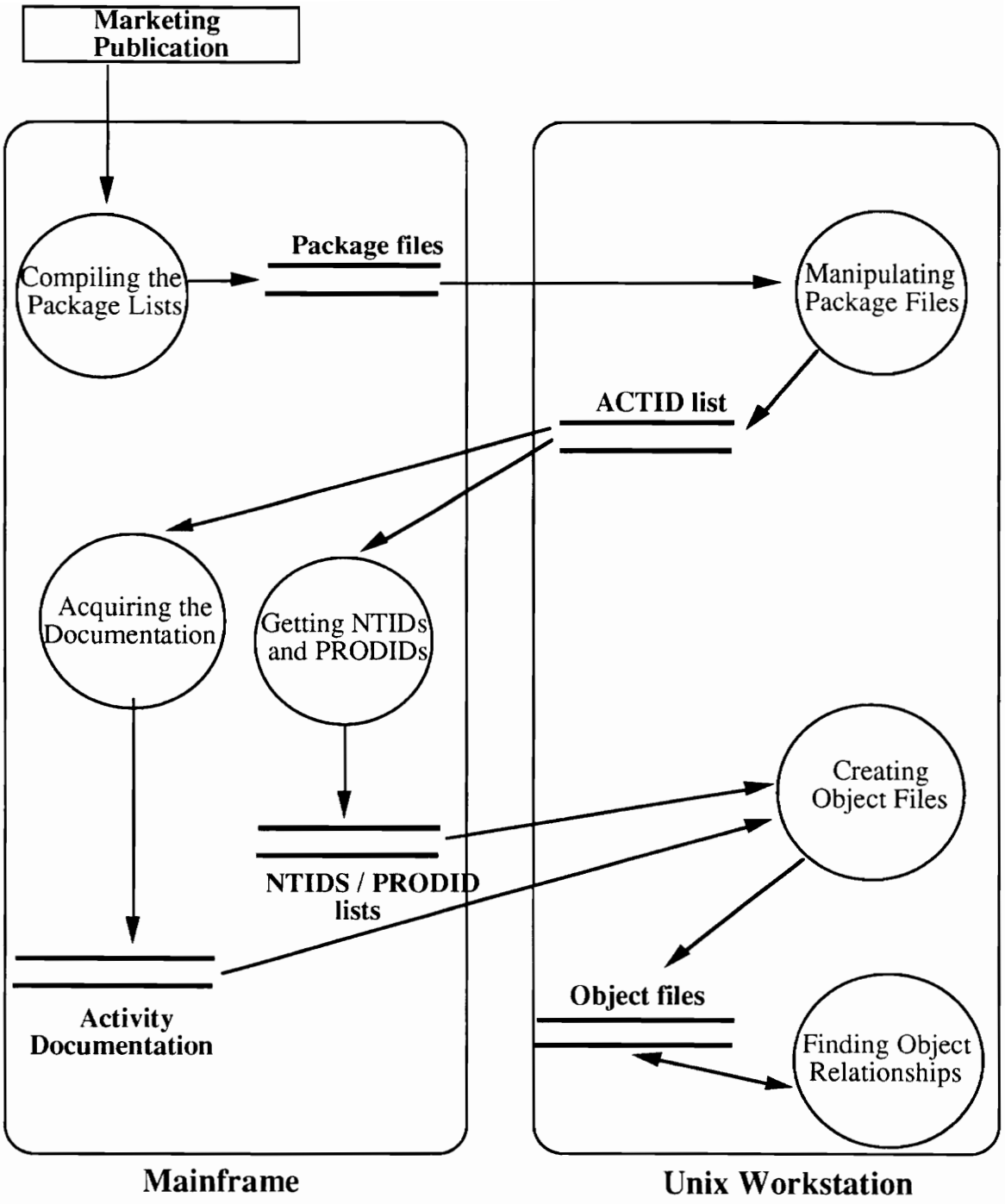

FIGURE 4.4 Knowledge Acquisition Data Flow Diagram

- The filename of each package list is modified, a list of all activities or ACTIDs is compiled and sent to a mainframe account that has access to activity documentation, i.e., an account that has access to all of the *PLS DOC* libraries and can link to the *DMSFILES* disk.

- On the mainframe account (using the list of activities), a search is conducted for activity documentation in the *PLS DOC*, *PLS OLDOC*, and *PLS OLDOC2* libraries, and on the disk *DMSFILES*. The activity documentation is then sent to a workstation.

- Next, the PRODID and NTID of each activity will be found. On a mainframe account that can link to the *PTATOOLS* disk, the PRODID for each ACTID or activity is found by using the *PKGVU* tool. On a mainframe account that has access to *FDB*, the NTID for each ACTID or activity is found.

- Each activity document will be made into a directory. For each activity, a number of sections of the corresponding activity document will be extracted and stored as separate files within the activity directory. The following sections will be extracted: interactions, references, restrictions/limitations, and synopsis. LISP object files are created for each activity. The release object files are created using known BCS dates.

- Activities will be grouped into feature families such as Call Waiting, etc. In addition, all activity interactions, related documents to an activity, and relatives of an activity will be found.

4.5 APMS Server

The Application Performance Monitoring System (APMS) is not a part of the SINK system. APMS is used to provide application usage information. SINK works with APMS so that SINK usage information can be stored there. For more information about APMS, see Appendix E.

CHAPTER FIVE

SINK USER INTERFACE

This chapter presents the general principles of interface design and guidelines that were used in the design of the user interface components. The rest of the chapter discusses the conventions that are common to all SINK user interface components (windows). For detailed description of the user's interaction with each of the components (windows) see Appendix D. The user interface for the SINK clients is implemented in the OSF/Motif toolkit under the X Window System.

5.1 Principles of Interface Design

During human-computer interaction, most information conveyed by the computer is presented visually, typically by means of a dynamic display device. No matter what display technology is used, the content and format of the information presented will always have a substantial impact on the usefulness and efficiency of the interaction. If the system does not present the information that the user needs, or presents it in an unusable or confusing manner, the user's ability to perform the necessary task may be sharply degraded. The *OSF/Motif Style Guide* lays down a set of guidelines that facilitate the design of applications that would increase user effectiveness and satisfaction [OSF 90a]. In designing the user interface for the SINK clients, the general principles laid down in the *OSF/Motif Style Guide* were followed.

The user interface for the client of the SINK system was designed to be easy to use and consistent by adopting the following [OSF 90a] :

1. Putting the user in control.
2. Adopting the user's perspective.
3. Using real-world metaphors.
4. Keeping the interface natural.
5. Communicating application actions to the user.

5.1.1 Putting the user in control

The user must have a sense of control over the application in order for him to use it effectively [THIM 90][FRES 87b]. Making users feel they are in control has two components, a cognitive component and a behavioral component. The cognitive component means that the user understands the structure of the software interface and can predict its response to a certain direction/command. The behavioral component means that users know what actions they need to perform to accomplish the tasks. An effective interface allows users to (1) form an accurate and detailed cognitive representation of the structure of the software and to (2) learn quickly how to operate or use it.

The sense of control can be given to the user by providing flexibility in accessing options [OSF 90a], direct manipulation, ensuring that the necessary and common functions are presented first and in a logical order [OSF 90a] and using dialogs to hide settings that are not accessed very often [OSF 90a]. Thus mnemonics were added to the Menu-bar options. The user can access the "FILE" menu on the main window by clicking on the cascade button labeled "FILE" or by using the mnemonic Alt-F. For options in the pull-down menus, accelerators are provided that allow keyboard access to functions in the pull-down menus.

Direct manipulation is so called because it manipulates objects in a very direct way [THIM 90][OSF 90a]. A number of operations such as deleting, duplicating, printing and editing may be done to any selected object directly. In the SINK interface there are a number of instances of direct manipulation. In getting the "activity synopsis", for example, the user selects an activity from the activity browser and then selects the activity synopsis from the tools menu. All these actions can be accomplished by moving the mouse sprite to the appropriate location and clicking, or by using accelerators. This paradigm reduces the amount of information that the user needs to memorize [OSF 90a].

The user must be able to re-configure the application without trouble [OSF 90a]. This enhances his sense of control over the application [OSF 90a]. This can be done by a mechanism built into the X Window System. The X resource manger reads the resource values for all the user interface objects from an ASCII resource file. The resource filename is the name of the application class [NYEA 88a] [NYEA 89b] defined in the application code.

The resource file for the SINK interface is called SINK. The X resource manager looks for the resource files in certain standard locations. If the UNIX environment variable XAPPLRESDIR is used to specify the full pathname of the directory that contains the X application resource files, then the files in this directory will be used. If not, then the X resource manager looks for the resource files in the user's home directory. If the files exist in the user's home directory, then the resource specifications from the files will be used. If the resource files do not exist in the user's home directory either, then the X resource manager will look in the default location namely:

/usr/lib/X11/app-defaults

If the resource files reside in this directory, then the resources specified in these files will be used. If the resource files do not exist here either, then the application will use default values for those resources that are not specified in the application code. The resources specified in the application code cannot be over-ridden from the resource files. This allows user customization without any need to re-compile the application.

5.1.2 Adopting the user's perspective

Since the problem at hand for the SINK system was to design an interface to access a feature interactions knowledge base, the actions the users (designers and testers) of such a system would like were studied informally and evaluated. The prototype user interface was demonstrated to various teams of potential users (telephony feature designers and testers). The feedback received from them in the form of comments, suggestions and additional requirements were incorporated in the final version of the user interface as appropriate. The various screens of the client of the SINK system are described in sections 5.2 - 5.14.

5.1.3 Using real-world metaphors

users are more comfortable when they can apply actions that they can draw analogies to in the real world [OSF 90a]. In daily life one is familiar with objects such a dial switches, radio buttons and light switches that toggle on and off. The user can relate to clicking on a push button to invoke actions as it is analogous to a real world situation. In the SINK system, the user interface screens were designed by using the interface components that would be most intuitive for the users such as push-buttons, scrolled windows and other user interface objects provided by the Motif toolkit.

users also expect rapid response from the system when a certain action is performed [THIM 90] [OSF 90a]. Response times that vary widely in the application result in the users getting confused because their train of thought gets interrupted. In the SINK system, care was taken to ensure that the response time was kept constant. This was done by following a lazy evaluation scheme [THIM 90]. In the lazy evaluation scheme the user interface objects (widgets) hierarchies needed for each of the screen components (windows) are built only when needed. The widget hierarchies can be created statically. However this would mean an increased start up time for the application. By creating the widget hierarchies dynamically, the start up time is spread more uniformly across the application. users would then see nearly constant time delays for the display of the various windows.

5.1.4 Keeping the interface natural

This consists of designing the application so that users may step through tasks in a manner natural to the problem at hand [OSF 90a]. A good interface design should minimize the need for the user to memorize and later recall information. Whenever possible, users should be able to choose from lists and be allowed to use their recognition memory rather than their recall memory. This was achieved in the SINK system by organizing the tasks in a logical and hierarchical manner. Thus the Main window lists the various browsers. The user starts from there and chooses one of the six browsers available. This ensures a natural flow for the task of information retrieval. Such a layout also provides for easy navigation [OSF 90a].

The user interface was designed such that there were no sharp contrasts between the objects on the screen and the background of the screen. When there are bright objects on a light background, for example, the user will be confused in terms of where he needs to focus [OSF 90a]. Too many objects that bear a sharp contrast to the background gives the application an unnatural feel [OSF 90a].

5.1.5 Communicating application actions to the user

The application must effectively communicate with the user by acknowledging the receipt of user commands, warning the user about certain actions, indicating to the user visually if certain actions will require some time for completion and displaying errors in a timely and concise manner [OSF 90a].

The SINK system was designed so that the application would provide the user with adequate feedback. OSF/Motif provides features such as highlighting the selected component of the menu option and indicating, by means of a "location cursor" when a component of the window has received focus. The location cursor is a rectangle that is drawn around the component such as a push-button.

When an error occurs, the error message is displayed back to the user via an ErrorDialog. The ErrorDialog displays a graphic symbol in addition to the message. This graphic symbol is a bitmap that is user customizable. In addition, the ErrorDialog causes the terminal to beep. This was done since it is known in general that users notice audio impulses better than they do visual impulses [THIM 90].

In order to display warning messages a WarningDialog has been provided. The user must acknowledge the warning message before he can proceed further with the application. Such dialogs are called modal. In Motif, modal dialogs may be application or system modal. In the case of application modal dialogs, the user may not proceed further in that application until the modal dialog has been dealt with. In the case of system modal dialogs, the user cannot process further in any application that is running on that system until the dialog has been dealt with.

In yet another class of dialogs, the user is queried about a certain action. The dialog that queries the user is called a QuestionDialog. The user, for example, may be asked the question "Are you sure?" in response to a request to delete a file. The use this class of dialogs is a required practice and will render the application less frustrating to use [OSF 90a].

5.2 Sample Session of the SINK System

The SINK user interface can be initiated by the user by simply typing "sink". The *main* window of the user-interface will pop-up on the display of the host machine. Transparent to the user a socket connection will be established with the Knowledge server and a shell script will startup the *APMS*. The *main* window or the very first screen that the user encounters upon invoking the SINK interface is shown in Figure 5.2. Once the socket con-

nection with the Knowledge Server has been successfully established the user can start "making requests" (using) the system.

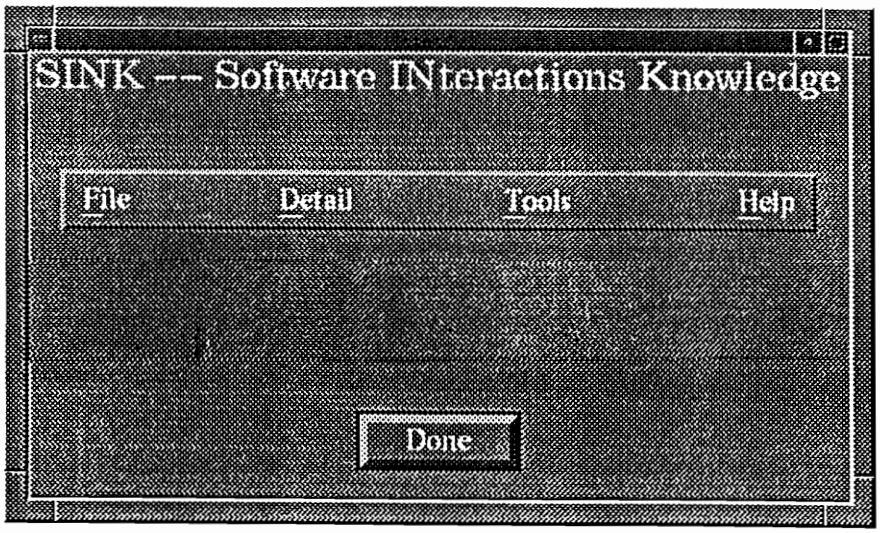

FIGURE 5.1 SINK Main Window

The main options available to the user are seen clearly in the menu bar that runs horizontally across the main window. The options that can be seen from this menu bar represent pull-down menus. This allows the designer of the interface to organize the actions and commands available to the user in categories such that these actions are functionally related to one another. The name of the category also reflects the broad range of functionality covered by the sub-options occurring under the category. For example, when the user selects the first option - "File" menu from the menu bar of the Main window using a mouse, he will see the sub-options - "Save", "Print", "Default Printer", and "Quit". These sub-options name the actions the user can perform. Thus the action word "File" is used to indicate to the user that this category is related to the actions affecting the whole file or session. By merely clicking on the menu bar option "File" the list of sub-options is displayed or *pulled down*. This allows the user to browse through the list of available options without being forced to actually select a particular action.

If the second option on the menu bar namely, "Details" is selected, then the sub-options of this menu item are displayed. The "Details" menu option displays sub-options which give more detailed information related to a window, thus indicating that "Details" refers to more extensive information about information already shown in the window.

The third option on the menu bar namely, "Tools", if selected from the menu bar of the Main window using a mouse, has the sub-options - "Activity Browser", "Feature Browser", "Package Browser", "Product Browser", "Release Browser", and "Undocumented Feature Browser". The action word "Tools" is used to indicate to the user that these are the different tools available to extract information from SINK. By merely clicking on the menu bar option "Tools" the list of sub-options is displayed or *pulled down*. This allows the user to browse through the list of available options without being forced to actually select a particular action.

The fourth option on the menu bar, namely "Utilities" has various sorting and searching choices which can be performed on the information present in the browser.

The last option on the menu bar is the "Help" which invokes the help index when selected and is the same for the windows in the SINK user interface. It is always the last option on the menu bar and is positioned on the far right end of the menu bar as specified in the OSF/Motif Style Guide [OSF 90a].

There also exists an option to quit out of the SINK interface by clicking on the "DONE" button at the bottom center of the Main window. This option is an instance of a particular type of user interface called a "Push Button". It is so named on account of its appearance and behavior. When the user clicks on this button, the button appears to be "pushed in" and simultaneously this pushing action alters the internal state of the user interface object and results in the invocation of a series of functions that were previously placed in a "callback" list to be invoked.

Let's assume the user selects the "Activity Browser" from the "Tools" menu of the Main window. The "Activity Browser" will pop-up on the screen as shown in Figure 5.2. This contains a list of activities with their respective ACTID's and the BCS, ProdID and

PackageID in which they are exist. The user can select any one activity at a time. All the windows which pop-up are modeless.

From this "Activity Browser" let's suppose the user selects the Acivity "CALLING_NAME_DISPLAY". To get more detailed information about this activity the user has to make a selection from the *Details* menu. Under the Detail menu, there are four options: *Synopsis*, *References/Related Documents*, *Interactions/Restrictions*, and *All* "Details". Selecting any of these three menus when an activity is highlighted pops-up a new window that displays the information.

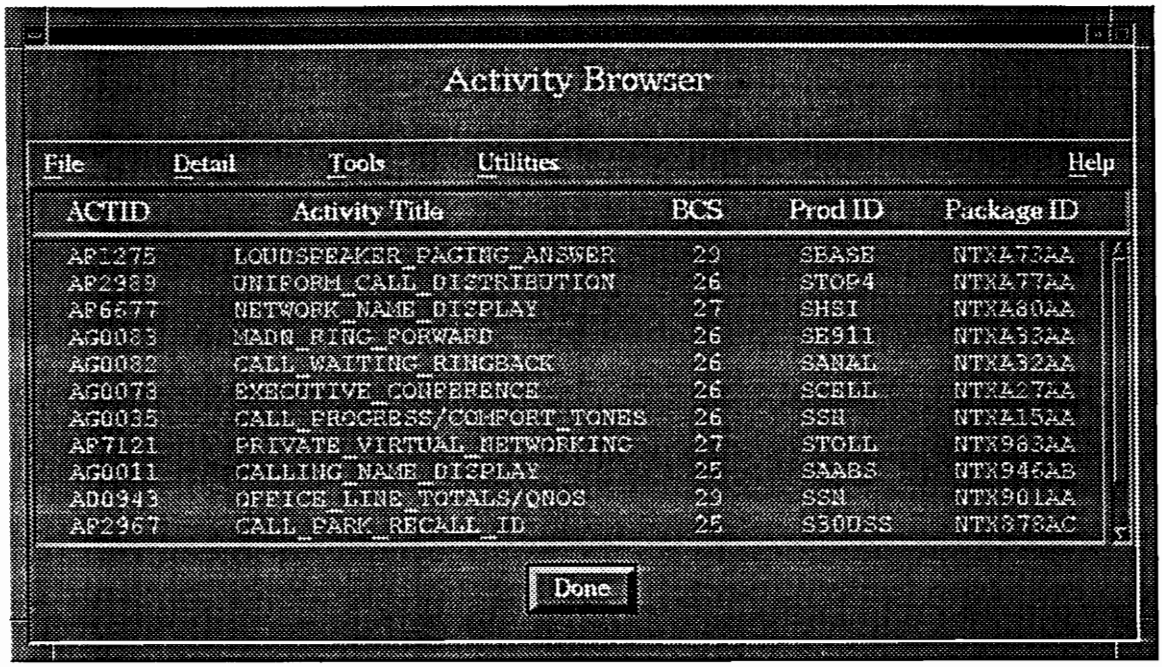

FIGURE 5.2 Activity Browser Window

If the user selects the "Synopsis" from this menu the "Activity Synopsis" window pops-up with the synopsis or summary of the activity "CALLING_NAME_DISPLAY". The activity synopsis window is shown in Figure 5.3. The user can select one of the "File"

menu options to save or print the information in the "Activity Synopsis" window. This user can select different activities and pop-up the "Activity Synopsis" window to see the synopsis, but the user can not pop-up two "Activity Synopsis" windows for the activity.

If the user selects the activity "CALLING_NAME_DISPLAY" and then selects the "References/Related Documents" option from the "Details" menu of the "Activity

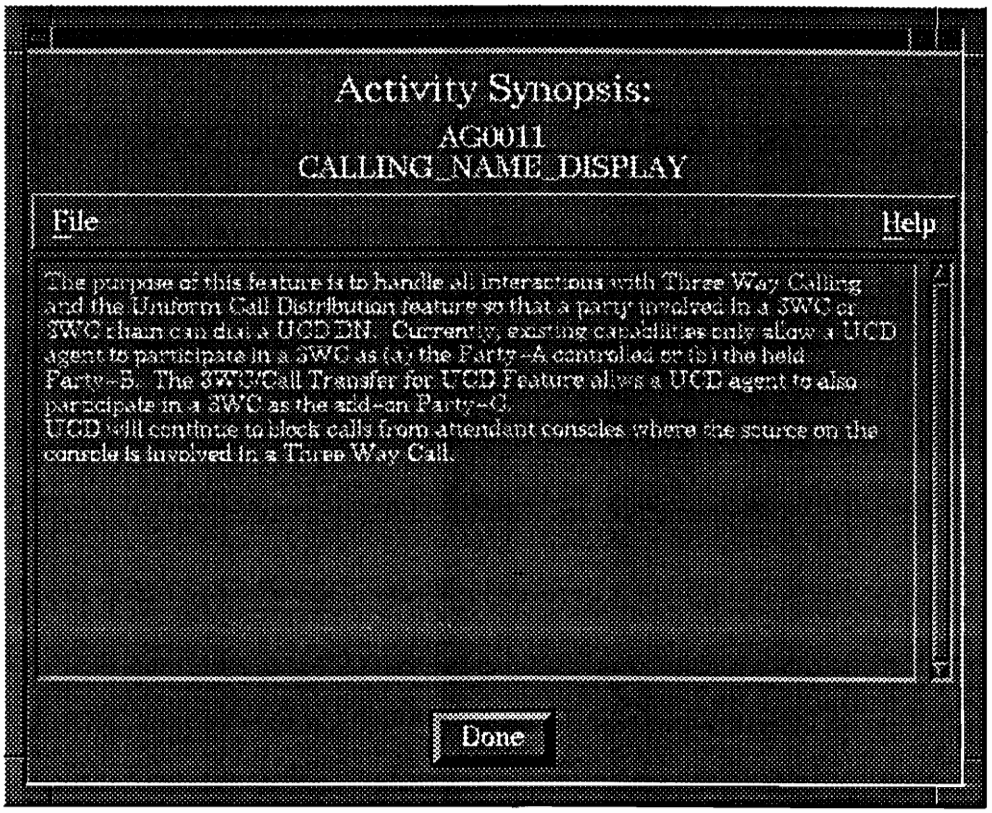

FIGURE 5.3 Activity Synopsis Window

Browser", the "Activity References" window will pop-up. This contains the list of activities which are mentioned in the reference section of the design document of this activity. It also shows the list of related documents discovered by the Knowledge Base.

The Activity References window is shown in Figure 5.4. The "File" menu is present here also as explained earlier.

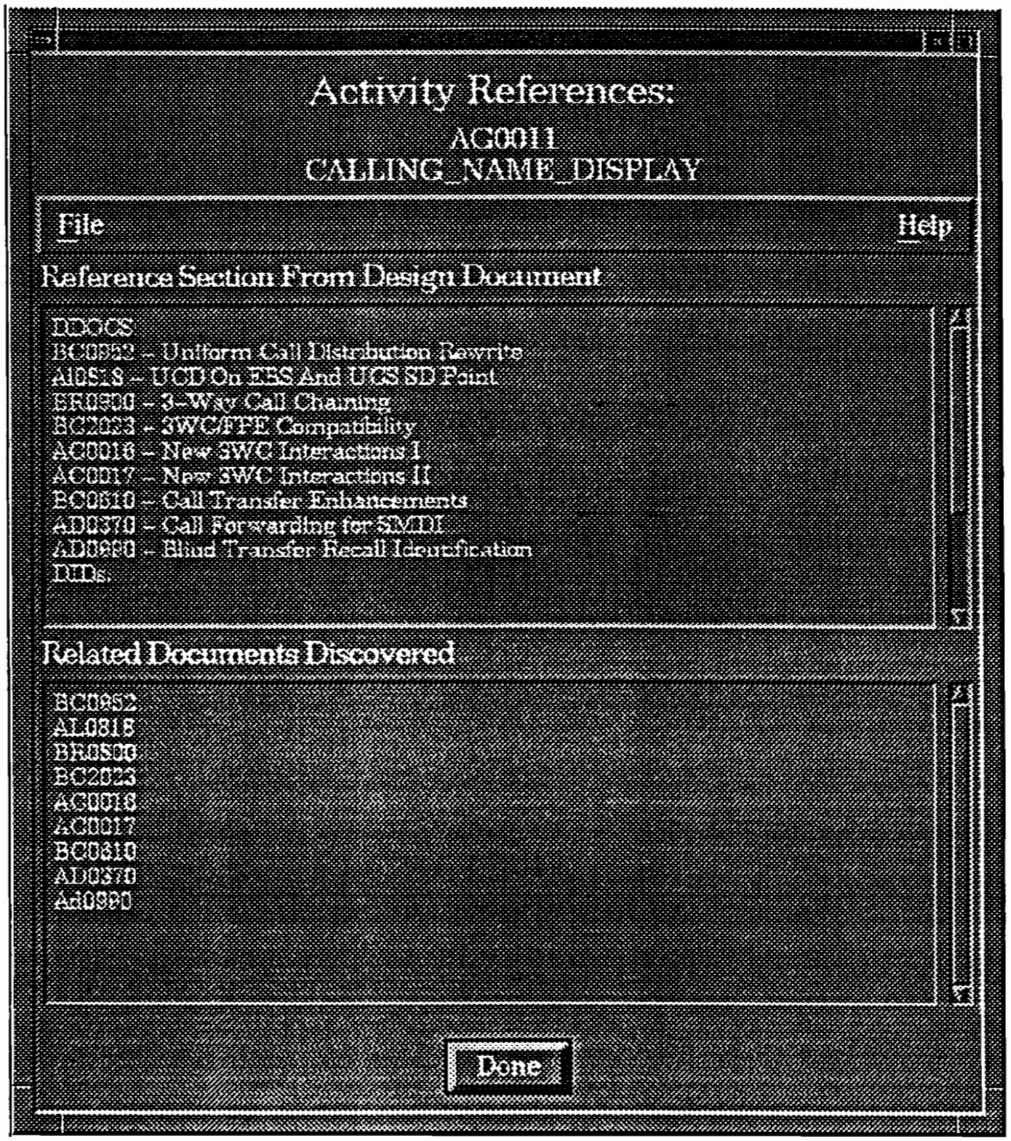

FIGURE 5.4 Activity References Window

If the user selects the activity "CALLING_NAME_DISPLAY" and then selects the "Interaction / Restrictions" option from the "Details" menu of the "Activity Browser", the "Activity Interactions" window will pop-up. This displays the Interactions and the Restrictions/Limitations section from the activity's design document, and the interacting features that were discovered by the knowledge Base. The Activity Interactions window is shown in Figure 5.5. The "File" menu is present here also as explained earlier.

For a detailed description of all the windows and the menu options and sub-options see Appendix D.

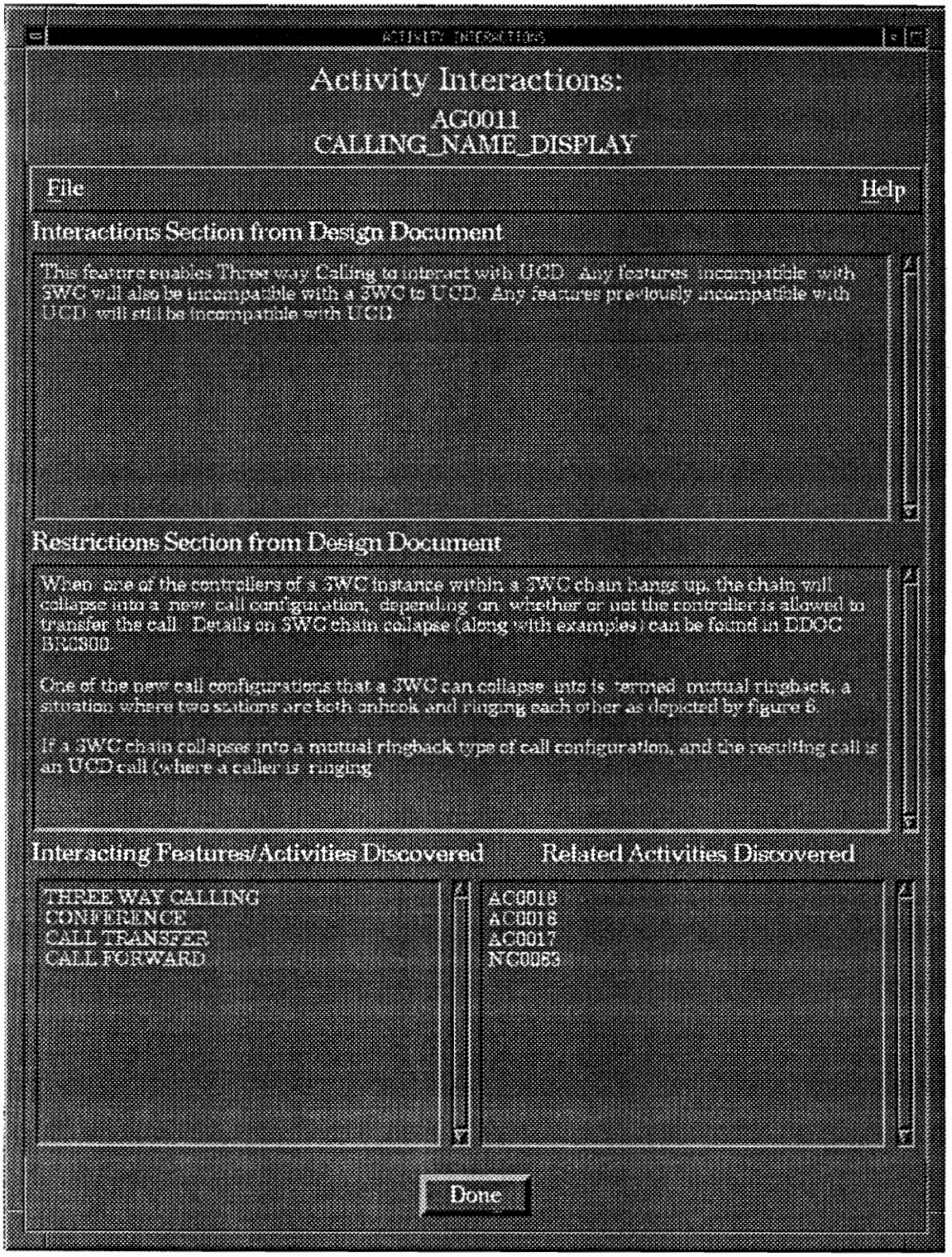

FIGURE 5.5 Activity Interactions Window

CHAPTER SIX

CLIENT-SERVER COMMUNICATION DATA FORMAT

This chapter describes the data format and methods used for communication between the SINK client and server. The objects of the SINK knowledge base have been explained in detail in Appendix C.

6.1 Request format

Every request sent from a Client to the Knowledge Server has the following format:

Client_Request(magic number, time-stamp, window-code, query)

where

- magic number is the socket communication port number that is known after a connection is established with the SINK Knowledge Server. It uniquely identifies the client.
- time-stamp tells the date and time when the query was generated.
- window-code is the identifier used by the interface manager to display the server response in the appropriate window.
- query is one of the queries shown in Tables 6.1.1 - 6.1.5. There are no end-of-line characters in any of the queries.

TABLE 6.1.1. List of activity queries

Query #	Query (Note: capitals denote variables)	Meaning of Query
a1	(activity-list-info)	List all activities and their corresponding BCS release, PRODID, and package ID.
a2	(activity-summary-info ACTID)	List the feature synopsis text of an activity's FN.
a3	(activity-interactions-info ACTID)	List the interactions section of an activity's FN and a list of interacting features.

a4	(activity-references-info ACTID)	List the references text of an activity's FN and a list of documents found in the text.
a5	(activity-all-info ACTID)	List the results of queries a2, a3, and a4 for the specified ACTID.

Note:
ACTID is an activity ID.
PRODID is a product ID.

TABLE 6.1.2. List of feature queries

Query #	Query (Note: capitals denote variables)	Meaning of Query
f1	(feature-list-info)	List all features and the corresponding number of activities in each.
f2	(feature-activity-list-info FEATID)	List the activities in the specified feature.
f3	(feature-interactions-info FEATID)	List the interacting features of the specified feature.
f4	(feature-references-info-FEATID)	List the references found in the activities of the specified feature.
f5	(feature-children FEATID)	List the activities in the specified feature.

Note:
FEATID is the name of a feature, such as "call_forwarding".

TABLE 6.1.3. List of package queries

Query #	Query (Note: capitals denote variables)	Meaning of Query
pkg1	(package-list-info)	List all packages and the corresponding number of activities in each.
pkg2	(package-activity-list-info PKGID)	List the activities in the specified package.
pkg3	(package-interactions-info PKGID)	List the interacting packages of the specified package.

pkg4	(package-references-info PKGID)	List the references found in the features in the specified package.

Note:
PKGID is a package ID.

TABLE 6.1.4. List of product queries

Query #	Query (Note: capitals denote variables)	Meaning of Query
prd1	(product-list-info)	List all products and the corresponding number of packages in each.
prd2	(product-activity-list-info PRODID)	List the activities in the specified product.
prd3	(product-interactions-info PRODID)	List the interacting products of the specified product.
prd4	(product-references-info PRODID)	List the references found in the activities of the specified product.
prd5	(product-children PRODID)	List the child products or packages that belong to a specified product.

Note:
PRODID is the name of a product, such as "CLASS".

TABLE 6.1.5. List of release queries

Query #	Query (Note: capitals denote variables)	Meaning of Query
rel1	(release-list-info)	List all releases and the corresponding number of packages in each of them.
rel2	(release-package-list-info RELID)	List the packages in the specified release.

Note:
RELID is the name of a release, such as "BCS31".

6.2 Reply format

A reply is sent from the Knowledge Server to a Client in either the form

Server_Reply(magic_number, time_stamp, window_code, result)

or

Server_Reply(error_message, window_code)

The first form reflects the successful processing of a client request. The "result" field is the information retrieved by the knowledge base for the query in the client's request. The formats of the results of every possible query are shown in Tables 6-10. New lines shown in the results have end-of-line characters preceding them.

The second form reflects that an error occurred in validating or processing a query. The error_message may be any character string.

TABLE 6.2.1. Format of activity query results

Query #	Query (Note: capitals denote variables)	Format of query results
a1	(activity-list-info)	(ACTID1 Activity-title1 BCS PRODID PACKAGE ACTID2 Activity-title2 BCS PRODID PACKAGE etc.)
a2	(activity-summary-info ACTID)	(ACTID Activity-title BCS PRODID PACKAGE "text-of-synopsis-section")
a3	(activity-interactions-info ACTID)	(ACTID Activity-title BCS PRODID PACKAGE "text-of-interactions-section" "text-of-restrictions-section" (list-of-interacting-features-and-activities))

a4	(activity-references-info ACTID)	(ACTID Activity-title BCS PRODID PACKAGE "text-of-references-section" (list-of-references-found-in-text))
a5	(activity-all-info ACTID)	(ACTID Activity-title BCS PRODID PACKAGE "text-of-synopsis-section" "text-of-interactions-section" "text-of-restrictions-section" (list-of-interacting-features-and-activities) "text-of-references-section" (list-of-references-found-in-text))

Note:
ACTID is an activity ID.

TABLE 6.2.2. Format of feature query results

Query #	Query (Note: capitals de-note variables)	Format of query results
f1	(feature-list-info)	(Feature-name1 #-activities-in-feature1 Feature-name2 #-activities-in-feature2 etc.)
f2	(feature-activity-list-info FEATID)	(ACTID1 Activity-title1 BCS PRODID PACKAGE ACTID2 Activity-title2 BCS PRODID PACKAGE etc.)
f3	(feature-interactions-info FEATID)	(Feature-name1 #-activities-in-feature1 Feature-name2 #-activities-in-feature2 etc.)
f4	(feature-references-info FEATID)	(REFERENCE1 REFERENCE2 etc.)

f5	(feature-children FEATID)	(ACTID1 Activity-title BCS PRODID PACKAGE ACTID2 Activity-title BCS PRODID PACKAGE etc.)

Note:
FEATID is the name of a feature, such as "call_forwarding".

TABLE 6.2.3. Format of package query results

Query #	Query (Note: capitals denote variables)	Format of query results
pkg1	(package-list-info)	(PKGID1 #-activities-in-package PKGID2 #-activities-in-package etc.)
pkg2	(package-activity-list-info PKGID)	(ACTID1 Activity-title1 BCS PRODID PACKAGE ACTID2 Activity-title2 BCS PRODID PACKAGE etc.)
pkg3	(package-interactions-info PKGID)	(PKGID1 #-activities-in-package PKGID2 #-activities-in-package etc.)
pkg4	(package-references-info PKGID)	(REFERENCE1 REFERENCE2 etc.)

Note:
PKGID is a package ID.

TABLE 6.2.4. Format of product query results

Query #	Query (Note: capitals de-note variables)	Format of query results
prd1	(product-list-info)	(PRODID1 #-features-in-product PRODID2 #-features-in-product etc.)

prd2	(product-activity-list-info PRODID)	(ACTID1 Activity-title1 BCS PRODID PACKAGE ACTID2 Activity-title2 BCS PRODID PACKAGE etc.)
prd3	(product-interactions-info PRODID)	(PRODID1 #-features-in-product PRODID2 #-features-in-product etc.)
prd4	(product-references-info PRODID)	(REFERENCE1 REFERENCE2 etc.)
prd5	(product-children PRODID)	two possible formats: (PRODID1 #-features-in-product PRODID2 #-features-in-product etc.)

Note:
PRODID is the name of a product, such as "CLASS".

TABLE 6.2.5. Format of release query results

Query #	Query (Note: capitals denote variables)	Format of query results
rel1	(release-list-info)	RELID1 #-features-in-release RELID2 #-features-in-release etc.)
rel2	(release-package-list-info RELID)	(PKGID1 #-activities-in-package PKGID2 #-activities-in-package etc.)

Note:
RELID is the name of a release, such as "BCS31".

CHAPTER SEVEN

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

The aim of this project was to design and develop the SINK (Software INteractions Knowledge) system based on the client server paradigm. A concurrent, stateless network oriented knowledge server has been developed, thus enabling users on the network to access the SINK knowledge server by running the SINK clients on their local computer systems. The clients have a Motif based graphical user interface.

The SINK system is based on the client server model. It enables the effective use of scarce computing resources since a powerful machine can be used as a retrieval server which can be accessed by multiple users on the network. It is easy to maintain the SINK system as it is a central repository of information which can be modified or updated as needed without being concerned about the consistency of data which would be the case if copies of the knowledge base resided on local machines of users. The SINK system could be made more efficient if multiple copies of the server were executed on different machines; this would allow the search load to be distributed amongst various server machines. The advantages of distributed processing would apply and could be exploited further by continuing work in this direction.

The project has indeed been most invigorating and instructive. It has been instructive in that the author has learned a great deal more about information retrieval systems and distributed processing in the context of present day computing technology. It has been invigorating since the author has spent long and satisfying hours in completing the project and working with a very dedicated team. A number of lessons have been learned which may well be passed on to future students or researchers.

The SINK system, while it has some very distinct advantages, does have some limitations. Even though the knowledge server is concurrent in nature and can support multiple users at the same time, the over-all system is not truly multi-user, as the knowledge base

can handle only a single user at a time. Also the method for detection of activity interactions could be improved by using natural language techniques. The knowledge base could be made more portable and efficient if implemented in an object oriented paradigm such as the **L**arge **E**xtended Object-oriented **N**etworked **D**atabase (LEND) system developed at VPI&SU.

The SINK knowledge server is presently a 4.3BSD socket based server. A server that is based on the Transport Layer Interface (TLI) [STEV 90] with the same functionality would be more flexible. TLI was introduced with Release 3.0 of System V in 1986. TLI provides an interface to the transport layer of the OSI model, and was modeled after the ISO Transport Service Definition. There is one deficiency in the release of TLI, and indeed with all the Release 3.x systems from AT&T: although they provide a networking interface (TLI), they don't supply a transport provider as part of the basic system. The networking software that is supplied, cu, UUCP, and RFS (Remote File System), can all use TLI if an appropriate transport provider is available. System V Release 4.0 is to contain a TCP/IP transport provider. 4.3BSD systems, on the other hand, provide not only a networking interface (sockets) but also provide the Internet protocols, the Unix protocols, and the XNS protocols.

The development of the graphical user interface was found to be one of the most challenging areas. Even with several iterations, writing a simple but functionally rich interface is not altogether a straightforward task. The author was fortunate in being able to learn from previous development efforts of graphical user interfaces for retrieval systems such as **CO**mposite **D**ocument **E**xpert **R**etrieval (CODER) system at VPI&SU. The user interface has not been formally evaluated by a usability study, but it was demonstrated to potential users (designers and testers) during all phases of development and their comments, suggestions and requirements were taken into consideration.

The graphical user interface should provide the users with greater flexibility and render the systems usable since it provides features such as user control, direct manipulation and effective user-application communication [OSF 90a]. Again, since the user interface is inherently suited to multi-tasking on account of the use of the windowing concept in concert with UNIX operating system, the user has the additional flexibility of continuing with other tasks while simultaneously searching for the information needed.

REFERENCES

[ASEN 90] Asente, P.J., and Swick, R.R., *X Window System Toolkit*, Digital Press, Bedford, MA, 1990.

[BARK 91] Barkakati, N., *X Window System Programming*, SAMS: A Division of Mcmillan Computer Publishing, Carmel, IN, 1991.

[BELK 87] Belkin, N.J., and Croft, W.B., "Retrieval Techniques", *Annual Review of Information Science and Technology*, Elsevier Publishers, Vol 22 (1987), pp 109-145.

[BIRR 84] Birrel, A.D., and Nelson, B.J., "Implementing Remote Procedure Calls", *ACM Transaction on Computer Systems*, vol.2, no.1, Feb 1984, pp 39-59.

[BORG 86] Borgman, C.L., "Why are Online Catalogs Hard to Use? Lessons learned from Information Retrieval Studies", *Journal of the American Society for Information Science*. vol 23 (1986), pp. 387-399.

[CLEV 66] Cleverdon, C.W., Mills, J., and Keen, M., *Factors determining the performance systems*, 2 vols, Cranfield, U.K., College of Aeronautics, 1966.

[CLEV 84] Cleverdon, C.W., "Optimizing convenient online access to bibliographic databases", *Information Services and Use*, 4, 1984, pp 37-47.

[CORB 90] Corbin, J.R., *The Art of Distributed Applications: Programming Techniques for Remote Procedure Calls*, Springer-Verlag, NY, 1990.

[COUL 88] Coulouris, G.F., and Dollimore, J., *Distributed Systems*, Addison Wesley, UK, 1988.

[FOX 83] Fox, E.A., "Some considerations for implementing the SMART information Retrieval System under UNIX", *Technical Report TR 83-560*. Ithaca, NY: Cornell University, Ithaca, NY, August 1983.

[FOX 87a] Fox, E.A., "Testing the Applicability of Intelligent Methods for Information Retrieval", *Information Services and Use*, (1987) 7:119-138.

[FOX 87b] Fox, E.A., "A Development of CODER system: A Testbed for Artificial Intelligence Methods in Information Retrieval", *Information Processing and Management*, 23(4), 1987, pp 341-347.

[FRES 87a] Frese, M., Ulich, E. and Dzida, W., *Psychological Issues of Human Computer Interaction in the Work Place*, Elsevier Science Publishers B.V. (North Holland) Amsterdam, 1987.

[FRES 87b] Frese, M. "A Theory of control and complexity: Implications for the Software Design and Integration of the Computer System into the Work Place, in *Psychological Issues of Human Computer Interaction in the Work Place*, Frese, M., Ulich, E and Dzida, W., Elsevier Science Publishers B.V. (North Holland) Amsterdam, 1987.

[GOUL 85] Gould, J.D., and Lewis, C., "Designing for usability: Key principles and what designers think", *Communications of the ACM*, 28, 3, March 1985.

[HCI 90] Panel Discussion, "HCI seen from the perspective of software developers", *Human-Computer Interactions - INTERACT '90*, Elsevier Science Publishers B.V (North Holland), 1990.

[HELC 91] Heller, D., *Motif Programming Guide*, Volume Six, O'Reilly & Associates, Inc., Sebastapol, CA, 1991.

[HOLC 85] Holocomb, R., and Tharp, A.L., "The Effect of Windows on Man-Machine Interfaces", Proceedings of the 1985 ACM Computer Science Conference - Agenda for Computing Research. The Challenge for Creativity. (March 1985), pp 12-14.

[IBUK 91] *IBUKI: Common Lisp Users Guide, I*BUKI, Los Altos, CA, 1987.

[JUNG 90] Junger, J., Bouma, G., and Letanoux. Ph., "Intelligent user interface for a conventional program", *Human-Computer Interactions - INTERACT '90*, Elsevier Science Publishers B.V (North Holland), 1990.

[LONG 91] Long, J.O., "Software Interactions Knowledge, A Proposal", Dept. 3D11, Bell-Northern Research Inc., Research Triangle Park, NC, 1991.

[LONG 92] Long, J.O., Naidu, A., Grebner, S., Resnick, A., and Schutz, E., "SINK Design Document", Dept. 3D11, Bell-Northern Research Inc., Research Triangle Park, NC, 1992.

[NYEA 88a] Nye, A., *Xlib Programming Manual Volume 1*, O'Reilly and Associates, Sebastpol, CA, 1988.

[NYEA 88b] Nye, A., *Xlib Reference Manual Volume 2*, O'Reilly and Associates, Sebastpol, CA, 1988.

[NYEA 89a] Nye, A. and O'Reilly, T., *X Toolkit Programming Manual Volume 4*, O'Reilly and Associates, Sebastpol, CA, 1989.

[NYEA 89b] Nye, A. and O'Reilly, T., *X Toolkit Reference Manual Volume 4*, O'Reilly and Associates, Sebastpol, CA, 1989.

[OSF 90a] Open Software Foundation, *OSF/Motif Style Guide*, Prentice Hall, Book Distribution Center, Route 59 at Brook Hill Drive, West Nyack, NY 10995, 1990.

[OSF 90b] Open Software Foundation, *OSF/Motif Programmer's Guide*, Prentice Hall, Book Distribution Center, Route 59 at Brook Hill Drive, West Nyack, NY 10995, 1990.

[OSF 90c] Open Software Foundation, *OSF/Motif User's Guide*, Prentice Hall, Book Distribution Center, Route 59 at Brook Hill Drive, West Nyack, NY 10995 1990.

[OSF 90d] Open Software Foundation, *Application Environment Specification (AES) User Environment*, Prentice Hall, Book Distribution Center, Route 59 at Brook Hill Drive, West Nyack, NY 10995 1990.

[OSF 90e] Open Software Foundation, *"The Distributed Computing Environment"*, OSF Technical Seminar, Dallas, Texas Feb 1991.

[RUBI 84] Rubenstein, R, and Hersh, H.M., T*he Human Factor - Designing Computer Systems for People*, Digital Press 1984.

[SALT 68] Salton, G., Lesk, M.J., "Computer evaluation of indexing and text processing", *Journal of the ACM*, 1968, 15(1), pp 8-36.

[SALT 73] Salton, G., Recent studies in automatic text analysis and document retrieval, *Journal of the ACM*, 1973, 20, pp 258-278.

[SALT 83] Salton, G., *Introduction to Modern Information Retrieval*, New York, NY: McGraw-Hill, 1983.

[SHNE 86] Schniederman, B., "Designing Menu Systems", *Journal of American Society for Information Science*, Vol. 37, No. 2, 1986, pp 57-70.

[STAN 86] Stanfill, C., and Brewster, K, "Parallel free-text search on the connection machine system", *Communication of the ACM, 1986*, 29(12), pp 1229-1239.

[STEV 90] Stevens, R., *UNIX Network Programming*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[STUA 91] Stuart, K.C., Robertson, G.G., and Jock, D.M., "The information visualizer, an information workspace", *Human Factors in Computing Systems*, CHI '91 conference proceedings, ACM Press, 1991.

[THIM 90] Thimbleby, H., *User Interface Design*, ACM Press, Addison Wesley, New-York, NY, 1990.

[THOM 84] Thomas J.C. and Schneider, M., (Editors) *Human Factors in Computer Systems*, Ablex Publishing Corporation, Norwood, New Jersey 07648, 1984.

[YOUN 89] Young, D., *The X Window System: Applications and Programming with Xt (OSF/Motif Edition)*, Prentice Hall, Englewood Cliffs, NJ, 1989.

APPENDIX A

TERMINOLOGY

ACTID - Activity ID

Activity - A software development activity; this represents the software and functionality corresponding to a specific activity ID (ACTID)

APMS - Automated Program Monitoring System, designed to keep usage information on BNR tools and applications

BCS - Batch change supplement. A software release for the DMS family of products

DDOC - Design Document

DID - Design Intent Document

DMSFILES - A disk that contains all available DIDs

FDB - Feature Database

Feature - A telephony capability or enhancement. Features are typically capabilities such as call forwarding, call waiting, etc. which are currently composed of numerous individual ACTID's.

Feature Interaction - The effects two features have on each other

FN - Function Description, a section in the DDOC

Ftp - File transfer protocol. This is a method for transferring data electronically between computer systems.

Interaction - See feature interaction

Knowledge Acquisition - The process of collecting data and information that will be used to build a knowledge base

Knowledge Base - A computer program that maintains facts (data) separately from control information (rules) and uses those facts to discover new facts and make decisions

MDC - Meridian Digital Centrex

Module - A unit of software code, usually at least 1000 lines of code

NTID - Northern Telecom ID

Package - A grouping of activities for marketing purposes; packages are composed of PLS subsystems or areas.

PRODID - Product ID

Product - A grouping of potentially marketable software packages with a definable, testable list of software dependencies. Products do not overlap since a software package cannot be in two products.

Protocol - A set of conventions or rules that govern the interactions of processes within a network. In SINK, a protocol is used by the Server and Clients to describe how they will exchange information and interpret that information.

PKGVU - A mainframe tool that allows the user to find package and product information

RTM - Ready to manufacture

Related Documents - All of the documents that are mentioned in the FN of an activity or all of the documents that are mentioned in the FN's of activities that belong to the feature

Related Features - Activities that belongs to the same feature

Release - Synonymous with BCS

SINK - Software INteractions Knowledge

Socket - An endpoint for communication between processes

TCP/IP - Transport Control Protocol/Internet protocol. This is a method of communication agreed upon by both the Server and Client in SINK.

APPENDIX B

SINK SOURCE MODULES

The SINK Client user interface consists of about 10000 lines of source code while the networking modules of SINK Knowledge Server consists of about 750 lines of code. A brief description of the source modules and the directory structure is given below for the benefit of the researcher who will work on this system in the future. The SINK Knowledge Base is about 13.3 MB in size at present and is rapidly growing.

All the files (source and executables) related to the SINK system reside in /bnr/tools/sink. Every SINK developer will be able to find the various SINK directories and files through the SINKDIR environment variable, i.e., $SINKDIR. The following is the SINK directory structure.

/bnr/tools/sink	This is the root directory under which all the SINK directories exist.
/RCS	Repository for the RCS (Revision Control System) files.
/bin	Source and executables for the SINK Client and Knowledge Server.
e.g., /bin/sink_init.c	All C and Lisp files start with "snk_".
/data	All data files.
/data/dids	Raw data files are separated into directories which has a separate file for each important section.

e.g.,

`/data/dids/af1234`	The activity "af1234" is a directory which has the following four files, each one having information about that section of the activity with activity ID "af1234".
`af1234/summary`	Summary or synopsis of the activity "af1234".
`af1234/interactions`	Names of activities interacting with the activity "af1234".
`af1234/references`	Names of the references and related documents of activity "af1234".
`af1234/restrictions`	Restrictions of the activity "af1234".
`/misc`	Miscellaneous data files, such as lists of packages, products, etc.
`/objs`	Files containing knowledge base objects to be loaded into the knowledge base.
`/objs/feature`	The five main directories for different type of objects are the following.
`/objs/parentfeatures`	
`/objs/release`	
`/objs/packages`	
`/objs/products`	

e.g.,
/objs/feature/ab9999

/objs/feature/ab9999/defs.lsp

/packages	The raw listing of packages the *SMS* are separated into directories.

e.g.,
/packages/ntx100aa

/plsdocs	Raw *PLS* document files which are separated into directories with a separate files for each BCS.

e.g.,
/plsdocs/ac9876

/plsdocs/ac9876/summary

/plsdocs/ac9876/interactions

/plsdocs/ac9876/references

/plsdocs/ac9876/restriction

/tmp	All the raw files go here first before going into ~/sink/data/dids, ~/sink/data/packages, ~/sink/data/plsdocs. All files that exist here should also be preceded by "snk_".
/doc	All SINK documents, including: KBFN - requirements document, KBDD - design document,

	KBTP - test plan, KBKAG - knowledge acquisition guide, and KBUG - users' guide
/ka_tools	Source code and executables for automated knowledge acquisition.
/tmp	The SINK executables in ~/sink/bin use this directory as a temporary directory.

APPENDIX C

OBJECTS IN SINK KNOWLEDGE BASE

This chapter describe the objects in the SINK knowledge base. Each of these objects represents actual software components. Attributes marked with an asterisk ('*') are for future expansion purposes.

C.1 Activity

TABLE C1. Activity Object Attributes

Attribute name	Type of data expected	Source of data
bcs	release object	SMS
interactions	string	PLS
interacting-phrases	string list	inference
interacting-activities	activity object list	inference
modules*	module object list	SMS
NT-ID	string	SMS
NT-package	package object	SMS
features	feature object list	inference
PRODID	product object	PKGVU
references	string	PLS
related-documents	string list	inference
related-activities	activity object list	inference
restrictions	string	PLS
summary	string	PLS
title	string	SMS
worklist	any	any

C.2 Feature

TABLE C2. Feature Object Attributes

Attribute name	Type of data expected	Source of data
acronyms	string list	marketing publications
interacting-activities	activity object list	inference
interacting-features	feature object list	inference
interacting-phrases	string list	inference
key-terms	string list	marketing publications
related-documents	string list	inference
activities	activity object list	inference
worklist	any	any

C.3 Module

There are currently no objects of this type. This type of object is included for expansion purposes. No attributes are currently defined.

C.4 Package

TABLE C3. Package Object Attributes

Attribute name	Type of data expected	Source of data
releases	release object list	SMS
activities	activity object list	SMS

C.5 Product

TABLE C4. Product Object Attributes

Attribute name	Type of data expected	Source of data
child-products	parent object list	Documentation
packages	package object list	PKGVU
parent-products	parent object list	Documentation
title	string	Documentation

C.6 Release

TABLE C5. Release Object Attributes

Attribute name	Type of data expected	Source of data
end-date	string (RTM date)	DATAVU
order	integer	BCS schedule
packages	package object list	inference
start-date	string (PI date)	DATAVU

APPENDIX D

SINK USER INTERFACE GUIDELINES

The SINK user interface can be started by simply typing "sink". The user interface executables reside in $SINKDIR/bin as shown in Appendix B. The makefile that resides in $SINKDIR/bin is used to compile the user interface code. If the application is compiled with the -DDEBUG flag turned on, a number of informational or debug messages will be written out to the screen. In order to run the application in a silent mode, turn the -DDEBUG mode off when compiling. If the version of OSF/Motif that is being used is 1.1 or less, then the -DMOTIF_1_0 flag needs to be set when compiling. If the version of OSF/Motif is 1.1 or higher, then the -DMOTIF_1_1 flag needs to be set when compiling. In addition, the flag -D_NO_PROTO flag must be specified as the interface does not support ANSI C function prototyping. Extreme care must be taken to ensure that the include files for X and OSF/Motif are for the appropriate versions of X and OSF/Motif libraries. When using Motif 1.0, the X11 Release 4 version of Intrinsics (libXt.a) should not be used. Instead, the Intrinsic library supplied by OSF must be used. However when using Motif 1.1 or higher, the X11 Release 4 Intrinsic version of Intrinsics must be used. For example, when compiling the source code when using Motif 1.0 libraries and debug option turned on, the compile command issued in the makefile is:

```
cc -DBSD -D_NO_PROTO -DMOTIF_1_0 -DDEBUG -g -c snk_main.c
```

When using the Motif 1.1 (or higher) libraries, the compile command is:

```
cc -DBSD -D_NO_PROTO -DMOTIF_1_1 -DDEBUG -g -c snk_main.c
```

The following sections explain the various browsers and windows in the SINK interface in detail for the benefit of the researcher who will work on enhancing the user interface. To show the consistency in the user interface all menu and sub-menu options have also

been explained and all enhancements or modifications should be done keeping the existing interface style guide in mind.

D.1 Window Conventions

All windows in SINK have several components in common:

1. Title - The title appears at the top of the window.

2. Menu bar - The menu-bar is the principal method through which the user operates the interface. In the SINK windows, most menu bars hold five different menus; each menu provides different functions to the user.

a) *File* - is used to print information or save it to a file. The File menu has four options:

- Save - allows the user to output the window information to a file. The user will be prompted for a file name.
- Print - allows the user to output the window information to a printer.
- Default Printer Name - allows the user to change the name of the printer.
- Quit - terminates the window. This is the same as the Done button at the bottom of each window. However, in the Main Window, both Quit and Done will terminate all windows.

b) *Details* - is used to display more detailed information related to the window.

c) *Tools* - is used to bring up any of the 6 tools included in SINK: Activity Browser, Feature Browser, Package Browser, Product Browser, Release Browser and Undocumented Feature Browser.

d) *Utilities* - is used to perform either searching or sorting on the current list of items in a browser.

e) *Help* - is used to provide window-sensitive help text for the user. This menu is always located on the right edge of the menu-bar.

3. Push buttons - These are typically used to execute an action or terminate a window. The most typical push buttons will be Done, which always closes a window, OK, which prints or saves information, and Cancel, which closes a window without executing a command.

D.2 Main Window

D.2.1 Synopsis

The Main Window is used to initiate the 6 tools available in SINK:

1. Activity Browser
2. Feature Browser
3. Package Browser
4. Product Browser
5. Release Browser
6. Undocumented Feature Browser

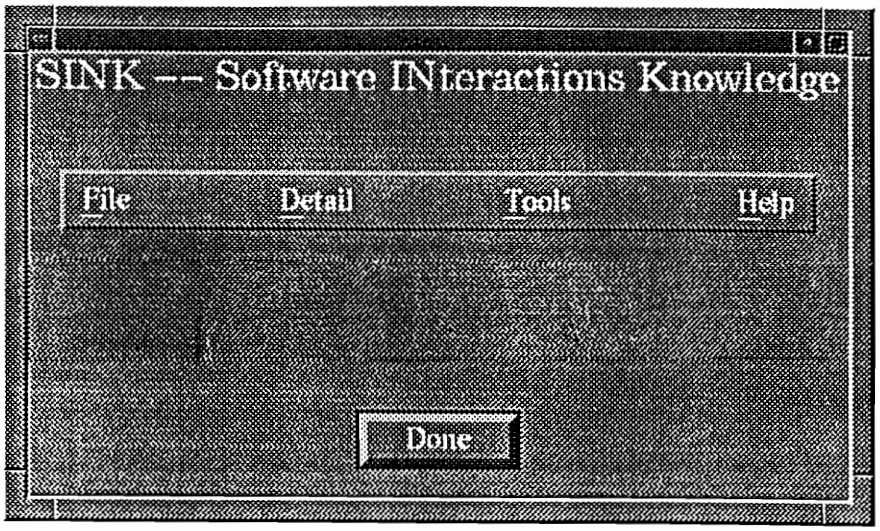

FIGURE D.1 Main Window

C.2..2 Initiation

When a user activates the SINK program, a connection will be established between the Client and Server and the Main Window will be displayed. This is the first screen the user will see upon executing the SINK program.

C.2.3 Functions Available

There are four choices on the menu-bar:

- The File menu is explained in Section D.1.
- Under the Detail menu, there is one selection: *Display Update Date*. This selection displays the date of the most recent update of the SINK knowledge base.
- The Tools menu holds six choices: *Activity Browser, Feature Browser, Package Browser, Release Browser, Product Browser, and Undocumented Interaction Browser*. Each of these items opens the corresponding browser.
- The Help menu holds only one choice: *Help Index*. This option causes the Help Index to be displayed. This window is explained in section D.12.

D.2.4 Termination

The Main Window is terminated by clicking the Done button or by selecting *Quit* from the File menu.

D.3 Activity Browser

D.3.1 Synopsis

The Activity Browser Window is used to view a list of all activities. This window is shown in Figure D.2. If the Activity Browser is selected from the Main Window, all activities are listed. Partial activity listings can be obtained from other Browsers, for example: If the user opens the Feature Browser and selects *List Child Activities* (from the Detail menu in the Feature Browser), a new Activity Browser would be opened listing only the activities in that feature.

D.3.2 Initiation

The Activity Browser Window may be initiated in two ways:

1. Selecting the Tools menu in the Main Window and selecting Activity Browser, or
2. Selecting Activities Browser from the Tools Menu of the Features Browser, the Packages Browser, the Product Browser, the Release Browser or the Undocumented Feature Browser.

D.3.3 Functions Available

In order to obtain information about a certain activity, the user selects an activity from the list displayed and selects a menu from the top menu-bar to obtain information. The menu-bar provides all of the options concerning activities:

- The File menu is explained in Section D.1.
- Under the Detail menu, there are four options: *Synopsis*, *References/Related Documents*, *Interactions/Restrictions*, and *All Details*. Selecting any of these three menus when an activity is highlighted opens a new window that displays the information.
- Under the Tools menu, there are five options: *Feature Browser* [see Figure D.6], *Release Browser* [see Figure D.7], *Package Browser* [see Figure D.8], *Product Browser* [see Figure D.9] and *Undocumented Feature Browser* [see Figure D.10].

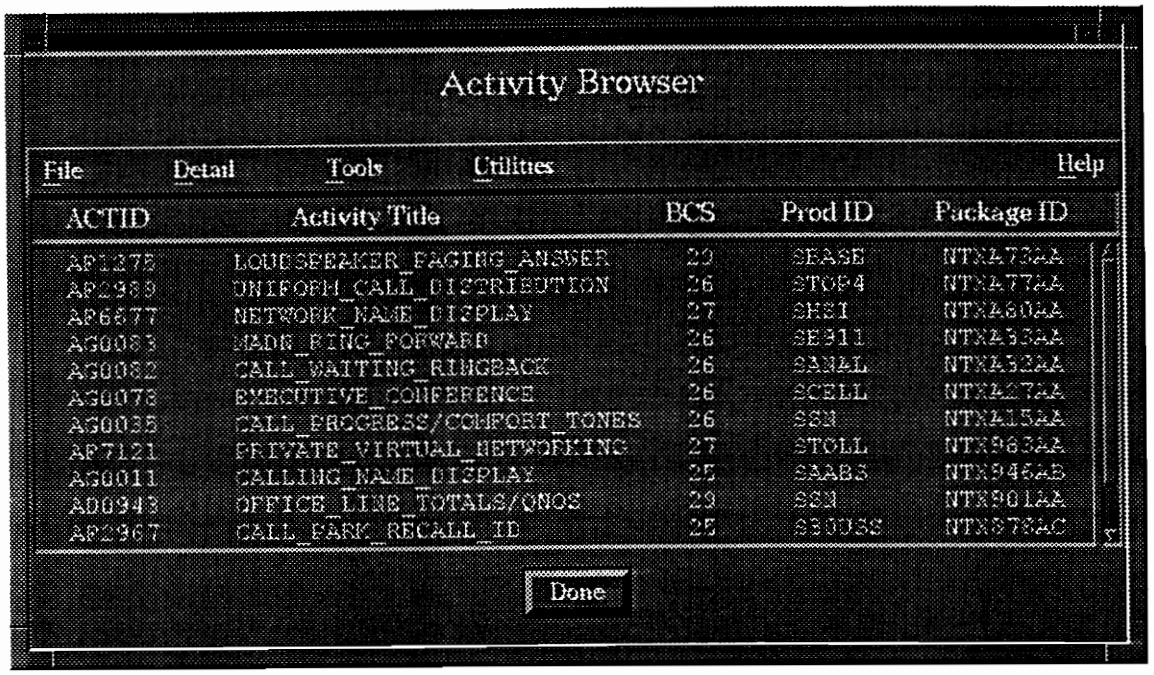

FIGURE D.2 Activity Browser Window

- Under the Utility menu, there are several options for searching and sorting: *Search ACTIDs, Search Activity Title, Sort by ACTID, Sort by Activity, Sort by Title, Sort by BCS, Sort by Prod ID and Sort by Package.* Under the search options, the user is prompted for a target to search for. If the item is located in the window, it is highlighted. If the item is not found, a warning message is displayed, telling the user that the search was unsuccessful.
- Under the Help menu, there is help for the File, Detail, Tools, and Utility menus and one more to activate the Help Index. If the user selects *Help Index*, the Help Index window will be displayed and the user can obtain help on any topic. If the user chooses any of the other specific Help options, the Help Index window will appear, but the Help Index will display information concerning the specific topic that the user selected from the Help menu. More information about help is in Section D.12.

D.3.4 Termination

This window is terminated by clicking on the Done button or by selecting the File menu and selecting *Quit.*

D.4 Activity Synopsis Window

D.4.1 Synopsis

The Activity Synopsis window, shown in Figure D.3, displays the Feature Synopsis section of the activity's FN or DID document.

D.4.2 Initiation

This window opens when the user clicks on either All Details or Synopsis from the Detail menu in the Activity Browser Window. All Details causes the Activity Synopsis, Activity Interactions and Activity References Windows to open, thus displaying all details about the activity. Selecting Synopsis causes only the Activity Synopsis Window to open.

D.4.3 Functions Available

The menu-bar has two menus:

- Under the File menu, there are four options: Save, Print, Default Printer Name and Quit. These functions are explained in Section D.1.
- Under the Help menu, there are several options: Help Index, Help on Saving, and Help on Printing. The Help Index is explained in Section D.12.

D.4.4 Termination

This window is terminated by clicking on the Done button or by selecting *Quit* from the File menu.

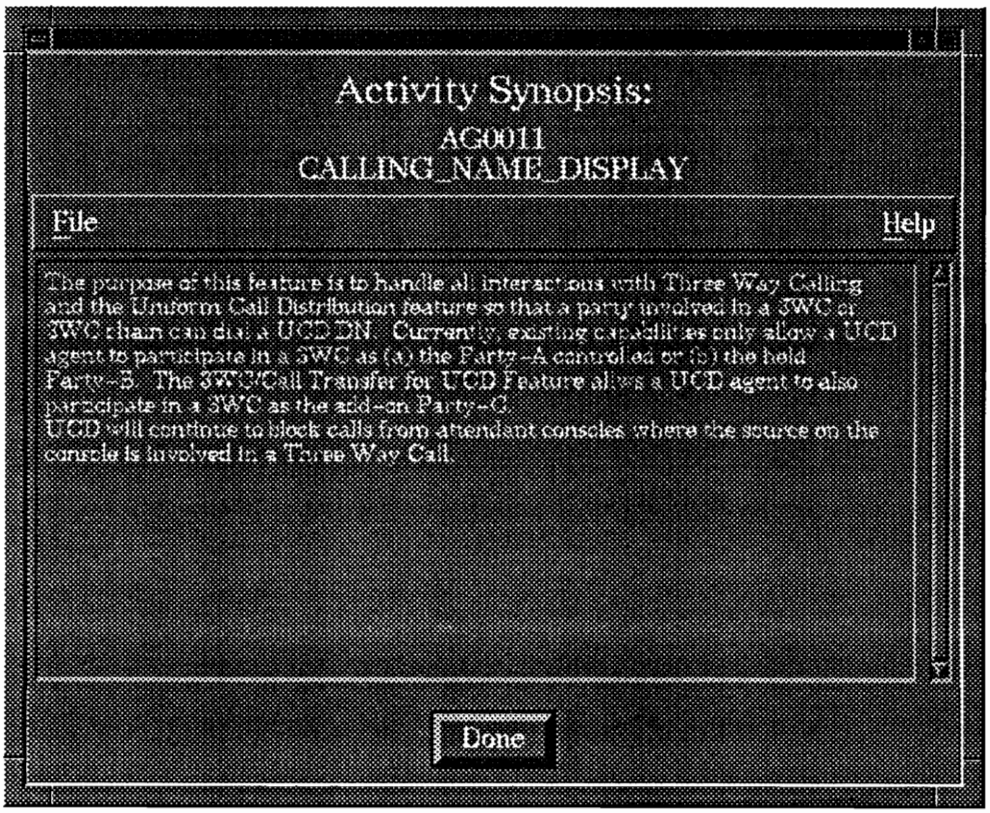

FIGURE D.3 Activity Synopsis Window

D.5 Activity Interactions Window

D.5.1 Synopsis

The Activity Interactions Window displays the Interactions section from the activity's FN, the Restrictions/Limitations section from the activity's FN, and the interacting features that were discovered by the knowledge base. This window is shown in Figure D.4.

D.5.2 Initiation

This window opens when the user clicks on either *All Details* or *Interactions* from the Detail menu in the Activity Browser Window. The *All Details* option causes the Activity Synopsis, Activity Interactions and Activity References Windows to open. Selecting *Interactions* opens only the Activity Interactions Window.

D.5.3 Functions Available

There are two menus on the menu-bar:

- Under the File menu, there are four options: Save, Print, Default Printer Name and Quit. These functions are explained in Section D.1.
- Under the Help menu, there are several options: Help Index, Help on Saving, and Help on Printing. The Help Index is explained in Section D.12.

D.5.4 Termination

This window is terminated by clicking on the Done button or by selecting *Quit* from the File menu.

D.6 Activity References Window

D.6.1 Synopsis

This window, shown in Figure D.5, displays the References section of the activity's FN and related documents discovered by the knowledge base.

D.6.2 Initiation

This window opens when the user clicks on either All Details or References/Related Documents from the Detail menu in the Activity Browser Window. The All Details option causes the Activity Synopsis, Activity Interactions and Activity References Windows

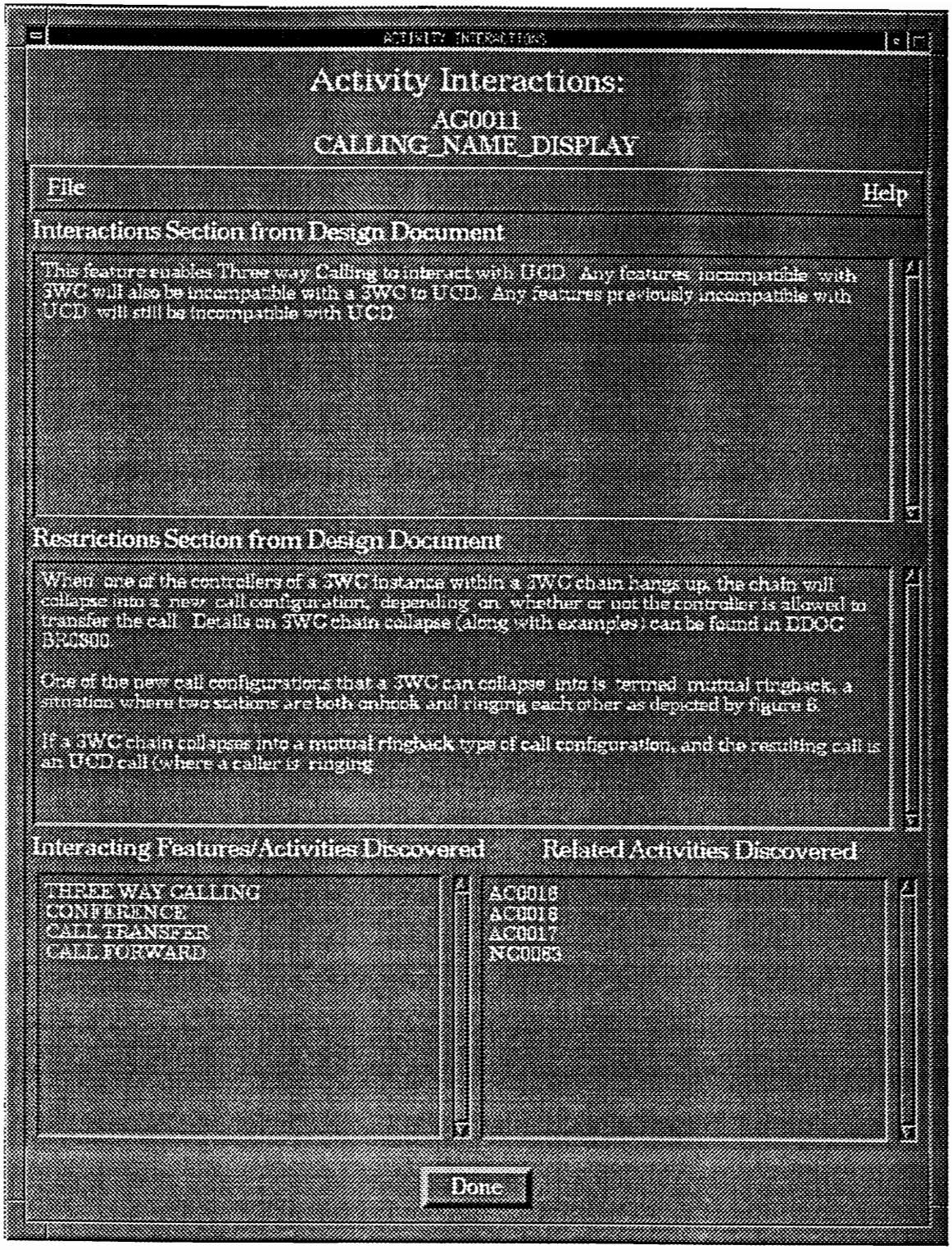

FIGURE D.4 Activity Interactions Window

to open. Selecting References/Related Documents from the Detail menu opens only the Activity References Window. The References section is displayed in the top part of the window and the related documents are displayed in the bottom part of the window.

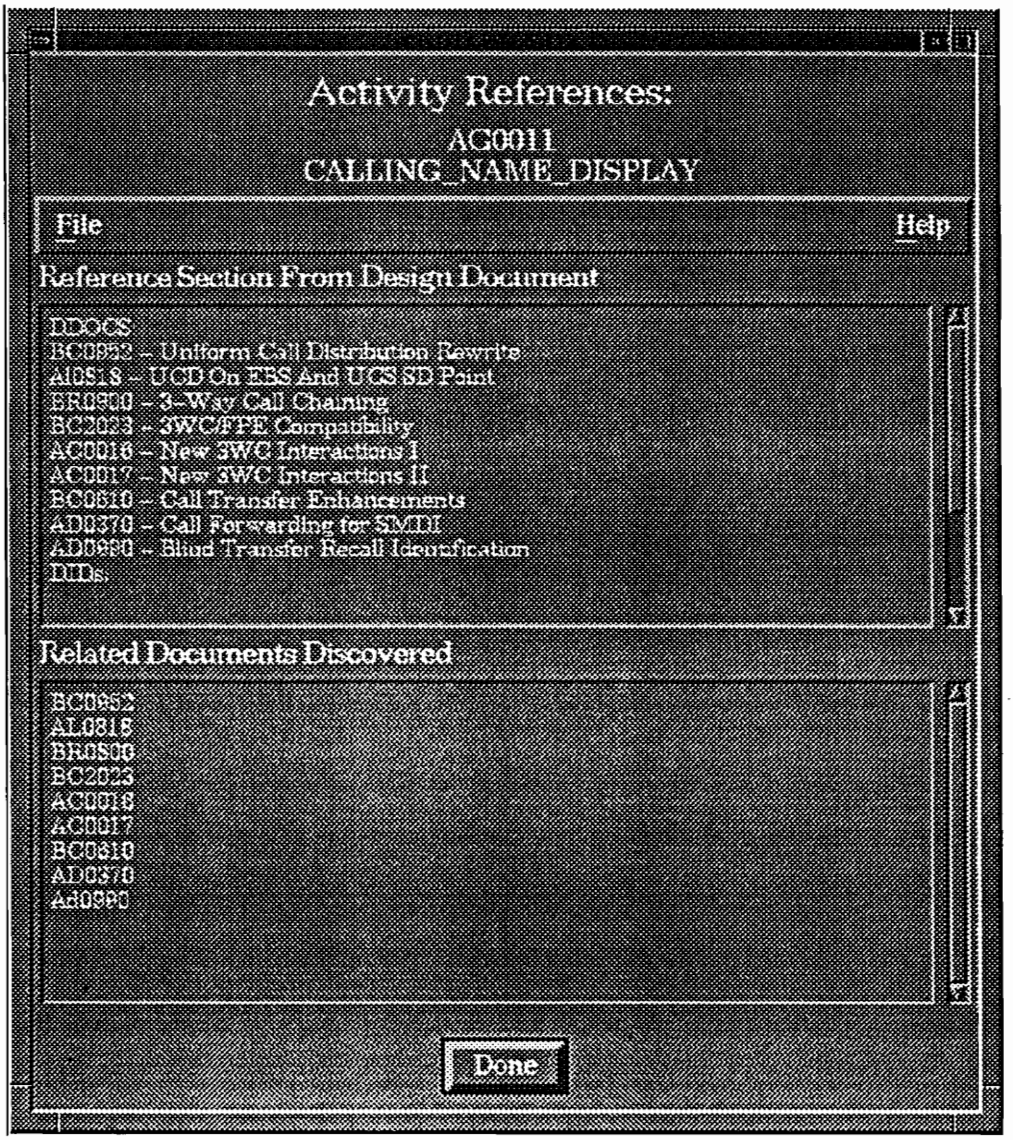

FIGURE D.5 Activity References Window

D.6.3 Functions Available

This window has two menus on its menu-bar:

- Under the File menu, there four options: Save, Print, Default Printer Name and Quit. These functions are explained in Section D.1.
- Under the Help menu, there are several options: Help Index, Help on Saving, and Help on Printing. The Help Index is explained in Section D.12.

D.6.4 Termination

This window is terminated by clicking on the Done button or by selecting *Quit* from the File menu.

D.7 Feature Browser Window

D.7.1 Synopsis

The Feature Browser Window is used to display a list of features. This window is shown in Figure D.6.

D.7.2 Initiation

The Feature Browser Window may be initiated in two ways:

1. Selecting the Tools menu in the Main Window and selecting Feature Browser, or
2. Selecting Feature Browser from the Tools Menu of the Activity Browser, the Package Browser, the Product Browser, the Release Browser or the Undocumented Feature Browser.

D.7.3 Functions Available

This screen has five menus on the menu-bar: File, Detail, Tools, Utilities and Help.

- The File menu is explained in Section D.1.
- The Detail menu has three choices: List Child Activities, List Interacting Activities, and List Related Documents. List Child Activities opens a new Activity Browser that displays only the child activities of the selected feature. List Interacting Activities opens a new Activity Browser that displays only the Activities that interact with this feature. List Related Documents opens a window similar to the Activity Synopsis Window, but documents related to this feature are displayed instead of an activity synopsis.

- The Tools menu has five choices: Activity Browser, Release Browser, Package Browser, Product Browser and Undocumented Feature Browser. Each of these options opens the corresponding browser.
- The Utilities menu has one option: Search Feature Titles. The user is prompted for a target to search for. If it is located in the Feature Names list in the Feature Browser, the target will be highlighted in the Window. If the search is unsuccessful, a System Message will be displayed to the user.
- The Help menu will display window-sensitive help text in the Help Index Window [see Section D.12].

D.7.4 Termination

This window is terminated by clicking on the Done button or by selecting *Quit* from the File menu.

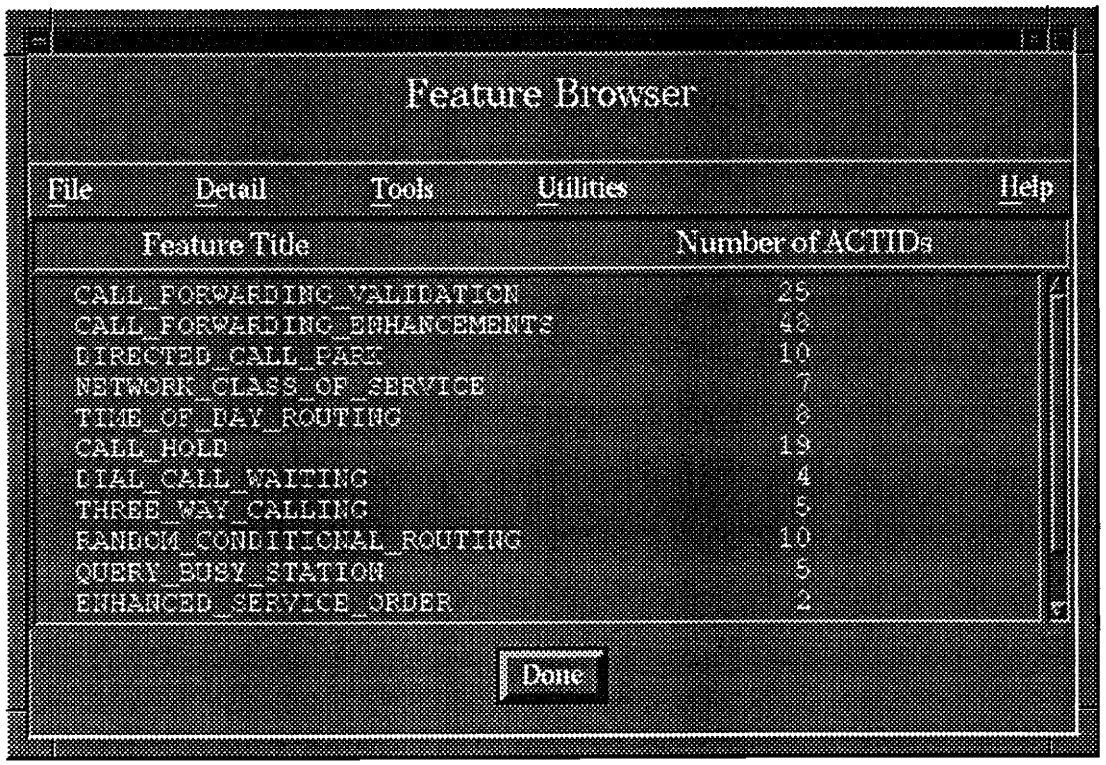

FIGURE D.6 Feature Browser Window

D.8 Release Browser Window

D.8.1 Synopsis

The Release Browser Window, shown in Figure D.7, is used to display a list of releases.

D.8.2 Initiation

The Release Browser Window may be initiated in two ways:

1. Pulling on the Tools menu in the Main Window and selecting Release Browser, or
2. Selecting Release Browser from the Tools Menu of the Activity Browser, the Feature Browser, the Package Browser, the Product Browser or the Undocumented Feature Browser.

D.8.3 Functions Available

There are five menus on the menu-bar: File, Detail, Tools, Utilities and Help.

- The File menu is explained in Section D.1.
- The Detail menu has one option: List Child Packages. The List Child Packages menu opens up a new Package Browser displaying the packages under the release selected from this Release Browser.
- The Tools menu has five options: Activity Browser, Feature Browser, Package Browser, Product Browser and Undocumented Feature Browser. Each of these options opens the corresponding browser.
- The Utilities menu holds one option: Search Release. If selected, the user is prompted for a target to search for. If it is located on the Releases Heading in the Release Browser, the target is highlighted in the Window. If the search is unsuccessful, a System Message is displayed to the user.
- The Help menu will display window-sensitive help text in the Help Index Window [See Section D.12].

D.8.4 Termination

This Window is terminated by clicking on the Done button or by selecting *Quit* from the File menu.

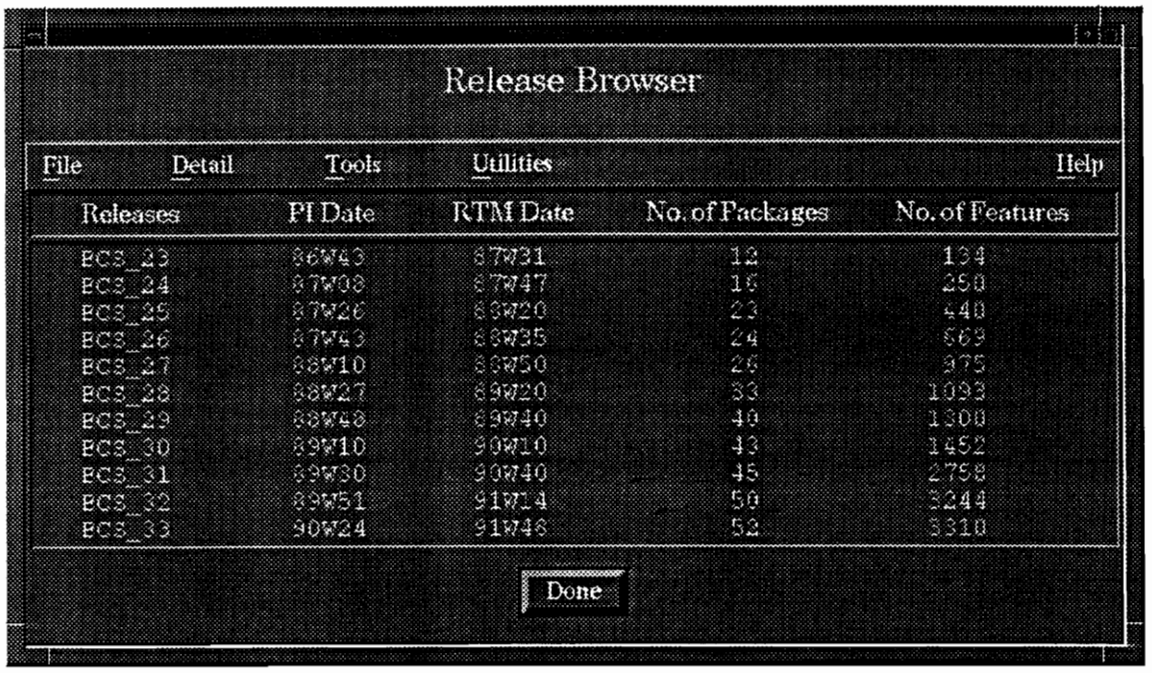

FIGURE D.7 Release Browser Window

D.9 Package Browser Window

D.9.1 Synopsis

The Package Browser Window, shown in Figure D.8, is used to display a list of packages.

D.9.2 Initiation

The Package Browser Window may be initiated in two ways:

1. Pulling on the Tools menu in the Main Window and selecting Package Browser, or

2. Selecting Package Browser from the Tools Menu of the Activity Browser, the Feature Browser, the Product Browser, the Release Browser or the Undocumented Feature Browser.

D.9.3 Functions Available

There are five menus on the menu-bar: File, Detail, Tools, Utilities and Help.

- The File menu is explained in Section D.1.
- The Detail menu holds two choices: List Child Activities and List Related Packages. List Child Activities opens up a new Activity Browser displaying only those activities in the selected package. List Related Packages opens up a Package Browser displaying only those packages related to the selected package.
- The Tools menu holds five options: Activity Browser, Feature Browser, Release Browser, Product Browser and Undocumented Feature Browser. Each of these options opens the corresponding browser.
- The Utilities menu has four options: Search Package ID, Search Package Titles, Sort by Package ID, and Sort by Package Title. If any of the Search options is selected, the user is prompted for a target to search for. If the search succeeds, the target is highlighted in the Window. If the search is unsuccessful, a System Message is displayed to the user.
- The Help menu will display window-sensitive help text in the Help Index Window [See Section D.12].

D.9.4 Termination

This window is terminated by clicking on the Done button or by selecting *Quit* from the File menu.

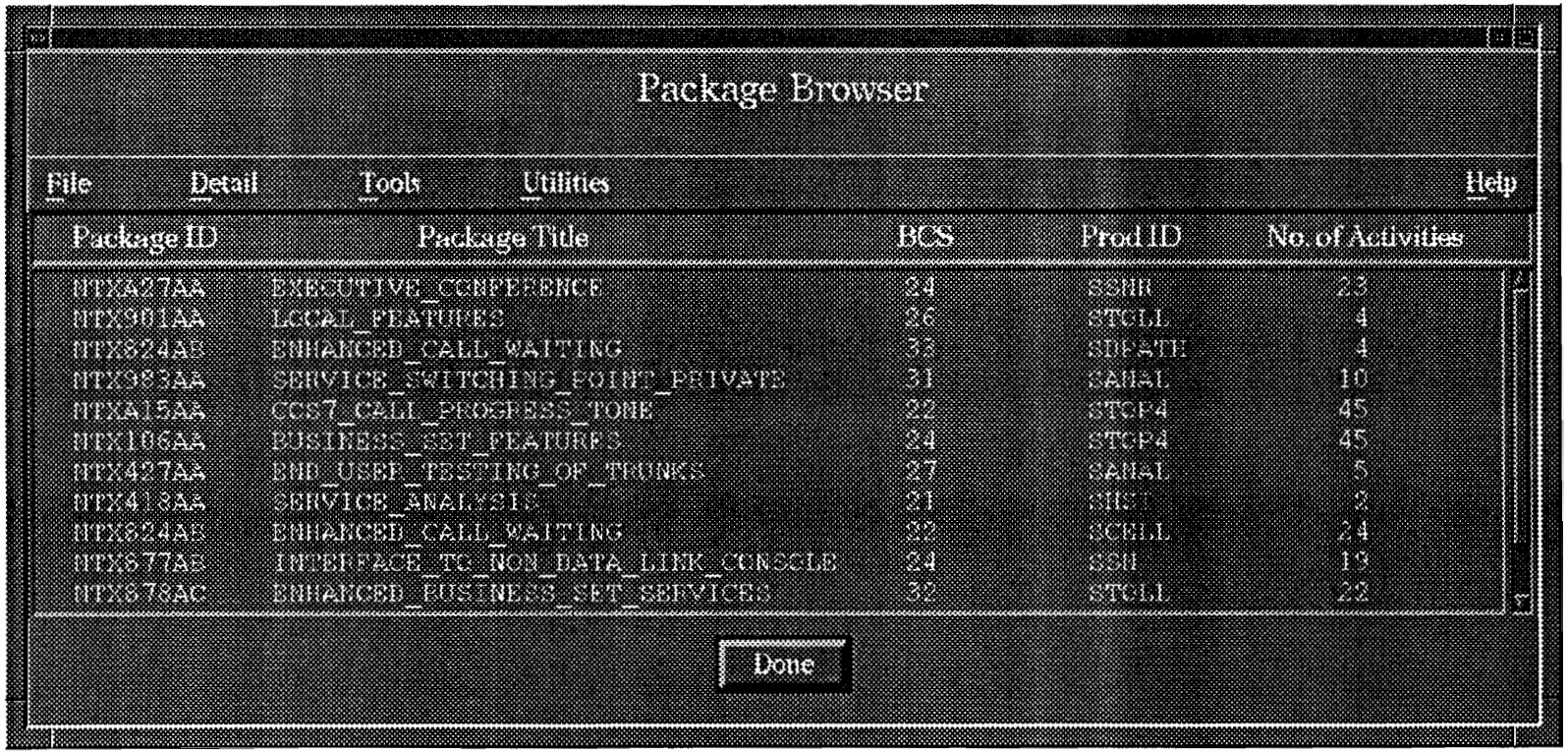

FIGURE D.8 Package Browser Window

D.10 Product Browser Window

D.10.1 Synopsis

The Product Browser Window, shown in Figure D.9, is used to display a list of products.

D.10.2 Initiation

The Product Browser Window may be initiated in two ways:

1. Pulling on the Tools menu in the Main Window and selecting Product Browser, or
2. Selecting Product Browser from the Tools Menu of the Activity Browser, the Feature Browser, the Package Browser, the Release Browser or the Undocumented Feature Browser.

D.10.3 Functions Available

This Window has five menus on the menu-bar: File, Detail, Tools, Utilities and Help.

- The File menu is explained in Section D.1.
- The Detail menu has three choices: List Related Products and List Child Packages and Display Product Map. List Related Products opens a new Product Browser, however, only related products are displayed. List Child Packages opens a new Package Browser displaying only those packages that interact with the selected product. Finally, Display Product Map opens a new screen displaying a map of all products.
- The Tools menu has five options: Activity Browser, Feature Browser, Release Browser, Package Browser and Undocumented Feature Browser. Each of these options opens the corresponding browser.
- The Utilities menu holds four options: Search Prod IDs, Search Product Titles, Sort by Prod ID and Sort by Product Titles. If the user selects anyone of the search options, the user is prompted for a target to search for. If the search is successful, the target is highlighted in the Window. If the search fails, a System Message is displayed to the user.
- The Help menu will display window-sensitive help text in the Help Index Window [See Section D.12].

D.10.4 Termination

This window is terminated by clicking on the Done button or by selecting *Quit* from the File menu.

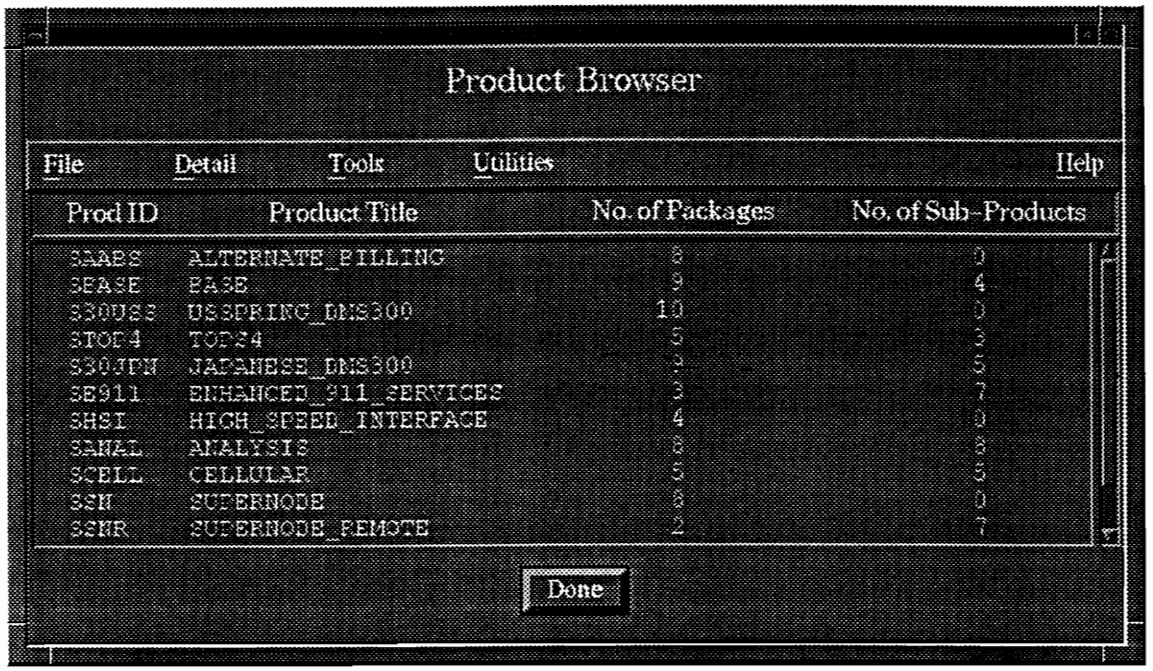

FIGURE D.9 Product Browser Window

D.11 Undocumented Interactions Browser

D.11.1 Synopsis

The Undocumented Interactions Browser allows the user to enter and record interacting features that have not been found or discovered by the Knowledge Server. This window is shown in Figure D.10.

D.11.2 Initiation

The Undocumented Interactions Browser may be initiated in two ways:

1. Pulling on the Tools menu in the Main Window and selecting Undocumented Interactions Browser, or
2. Selecting Undocumented Interactions Browser from the Tools Menu of the Activity Browser, the Feature Browser, the Package Browser, the Release Browser or the Product Browser.

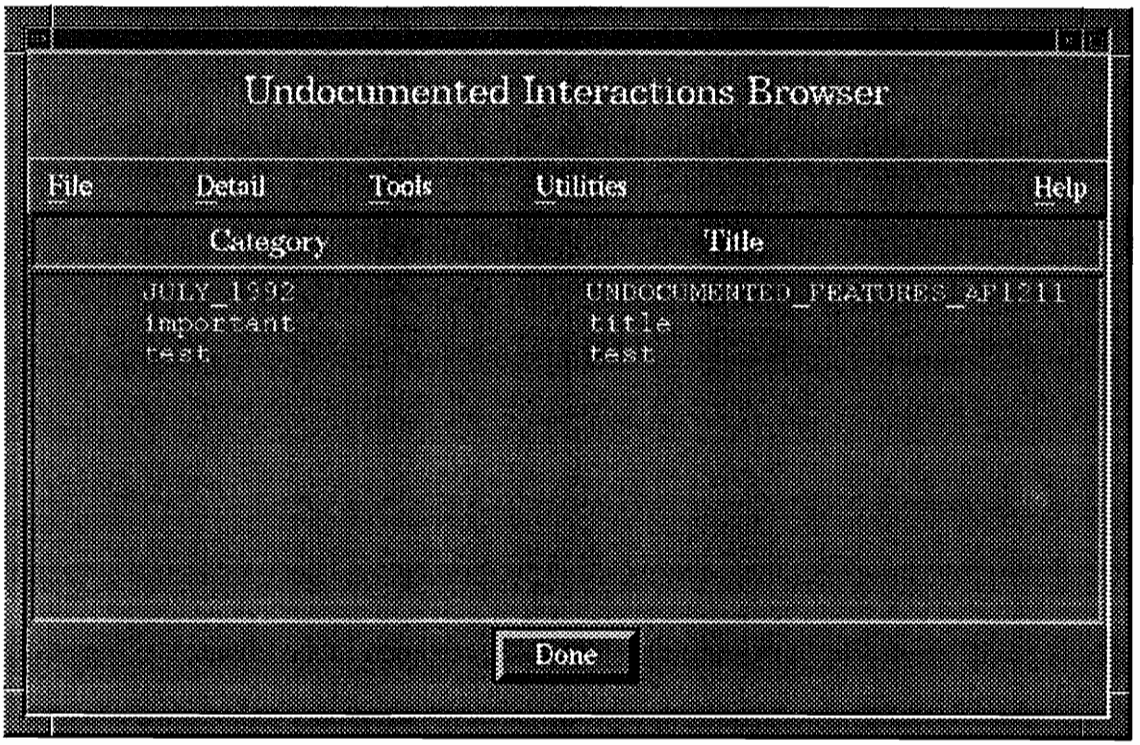

FIGURE D.10 Undocumented Interactions Browser

D.11.3 Functions Available

There are five menus on the menu-bar: File, Detail, Tools, Utilities and Help.

- The File menu is explained in Section D.1.

- The Detail menu holds three choices: Add Item, Edit Item and Display Item. Add Item opens up a New Undocumented Interactions Window that prompts the user for a category name, title name and interaction description. Edit Item opens up a Display Undocumented Interactions Window, however, instead of entering new information, the user has the option of changing existing information. Finally, Display Item opens up a window similar to the previous windows, however, the user is not allowed to make changes to the information displayed.
- The Tools menu has five options: Activity Browser, Feature Browser, Release Browser, Product Browser and Package Browser. Each of these options opens the corresponding browser.
- The Utilities menu holds four options: Search Category, Search Title, Sort Categories and Sort Titles. If the user selects any of the search options, he is prompted for a target to search for. If the search is successful, the target is highlighted in the Window. If the search fails, a System Message is displayed.
- The Help menu will display window-sensitive help text in the Help Index Window [see Section D.12].

D.11.4 Termination

This window is terminated by clicking on the Done button or by selecting *Quit* from the File menu.

D.12 Help Index Window

The Help menu can always be found on any menu-bar. It is the rightmost menu on the menu-bar. To obtain help, the user would select the Help menu from any menu-bar and select a specific topic. Subsequently, the Help Index Window opens and displays help concerning the topic selected. Or, the user can select Help Index and get help on any of the other topics listed. To get help on another topic, the user simply clicks on the topic, and the help is displayed on the screen on the right side of the window. The Help Index Window is shown in Figure D.11.

To Terminate or close the window, the user would click on the Done button and the window is removed from the screen.

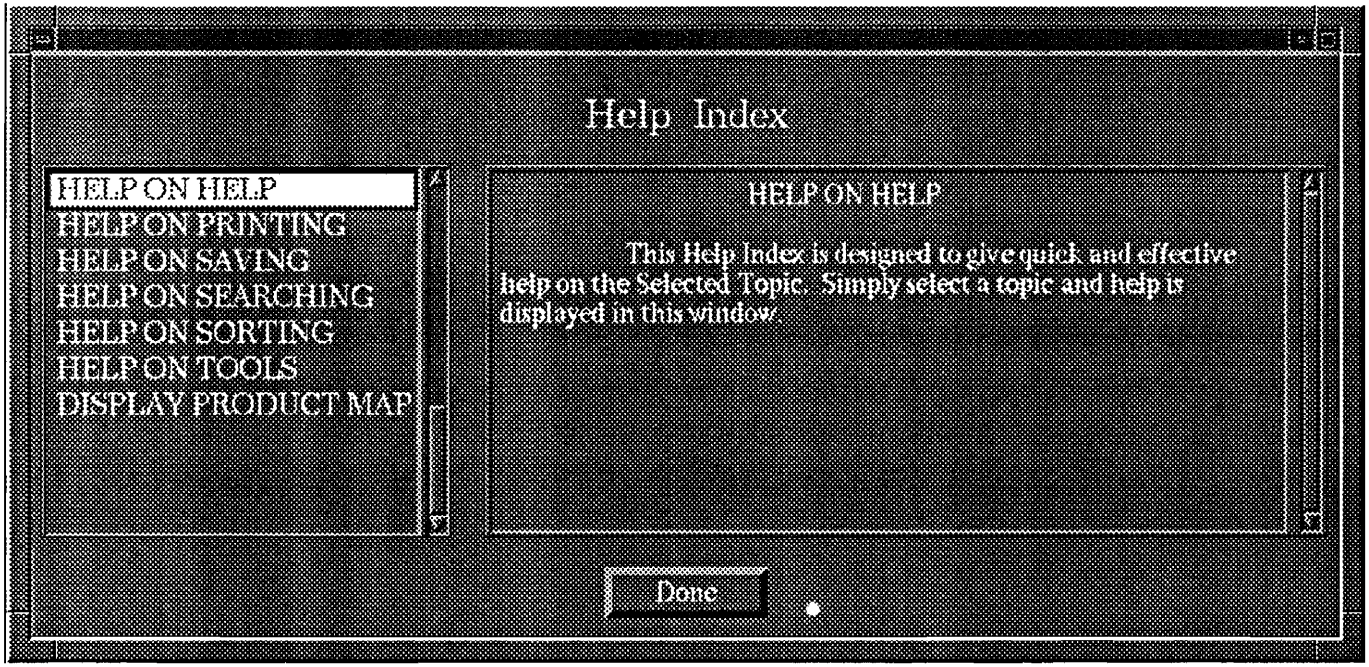

Figure D.11 Help Index Window

D.13 System Message Windows

D.13.1 Synopsis

System Messages are used to tell the user that a problem may have taken place during normal system processing or while processing a specific user-request. There are two types of system messages that may be displayed to the user:

1. Warning messages - These indicate problems that are not serious. Typically, a warning message indicates that a specific request finished execution, but there was some non-fatal problem that occurred.
2. Error messages - These indicate serious problems. An error message means that a specific request failed to execute or that a normal system operation is unable to be performed.

D.13.2 Initiation

System message are displayed under many circumstances, but the most typical reasons for displaying a System Message are to tell the user that an attempt to write to a file failed, an attempt to a print to a printer failed or that the user has placed too many requests and the Server is not responding fast enough.

D.13.3 Termination

System Messages are terminated by clicking on the Done button.

APPENDIX E

APPLICATION PERFORMANCE MONITORING SYSTEM

The Application Performance Monitoring System (APMS) is a system designed to provide application monitoring on the workstations in BNR. It provides centralized monitoring of applications running on all platforms including HP, Sun, and Apollo workstations. APMS requires monitored applications to send log entries to a designated server which receives, formats, and stores the information. These entries are processed regularly to produce reports detailing system usage.
There are three main components to APMS:

- APMS collection package
- APMS transport vehicle
- APMS reduction software

The APMS collection package provides two methods for applications to collect usage data. One is through a shell script, which is used when it is not possible or desirable to modify the source code of the monitored application. The shell script invokes the desired application, then calls the Unix *time* function to calculate the usage data. This is forwarded to the APMS server by the *apmssend* command. The second method is via a C language subroutine call that accomplishes the same objective.

The APMS transport vehicle employs a client/server approach. The client is invoked from the application whenever a log entry is required, typically after the application has completed. The client communicates with the APMS server via a TCP/IP connection. Once the connection is established, the client sends its data to the server, which writes it to a file. The APMS reduction software consists of a set of SAS programs which process the log entries received by the server and produce the reports.

The Computer Systems Management Group maintains an APMS server for application monitoring. This server is available to anyone in Information Technology wishing to

make use of the APMS facility. This server, which runs on the IT file-server, creates the log files and the standard set of APMS reports daily. Applications not using this server must define and install their own server. SINK uses the shell script method to record invocation data. Each SINK client session is initiated by calling a provided C-shell script as detailed above.