

Personal Communications

by

Chi Chong Wong

B.Sc., Systems Design Engineering

University of Waterloo

Waterloo, Canada

1989

SUBMITTED TO THE MEDIA ARTS AND SCIENCES SECTION, SCHOOL OF ARCHITECTURE AND
PLANNING, IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE DEGREE OF
MASTER OF SCIENCE

AT THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 1991

©Massachusetts Institute of Technology 1991

All Rights Reserved

Signature of the Author



Media Arts and Sciences Section


May 10, 1991

Certified by

Chris Schmandt
Principal Research Scientist

Thesis Supervisor

Accepted By



Stephen A. Benton

Chairman

Departmental Committee of Graduate Students

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUL 23 1991

LIBRARIES

ROTC

Personal Communications

by

Chi Chong Wong

Submitted to the Media Arts and Sciences Section, School of Architecture and Planning, on May 10, 1991 in partial fulfillment of the requirements of the degree of Master of Science at the Massachusetts Institute of Technology

Abstract

There is an emerging trend in the telecommunications industry for providing customizable communications services. Telephony features such as *call forwarding*, *call transfer*, *call waiting*, *hold*, *conference*, *voice mail* and *ring again* have become ubiquitous in the office environment. However, these telephony features have greatly increased the complexity of the central office (CO) switch and the telephone set.

One approach to managing complexity in the CO and providing a better user interface to advanced telecommunications services is to define a basic set of functions within the CO and to allow external peer entities such as workstations to interface to it through an open architecture. The integration of telecommunications systems into computer networks allows designers to shift complexity away from the switch by distributing the burden of call processing to a customer premise workstation. Conducting call management at the customer premise will allow the workstation to actively participate in call processing through intelligent agents.

A *phoneserver* that provides an architecture to controlling telecommunications services provided by an AT&T ISDN 5ESS switch is presented in this thesis. An *automated call management entity* (ACME) that interfaces to the phoneserver on a client's behalf is presented, along with other network-based voice services that utilize the phoneserver. ACME is a rule-based system that is configured using a graphical user interface called Phoneditor. It was found that it is a non-trivial task to define a set of rules that could adequately specify the personalization needs of users. These findings are discussed along with conjectures about the utility of a phoneserver and an ACME in advanced telecommunications networks of tomorrow.

Thesis Supervisor: Chris Schmandt

Title: Principal Research Scientist

Contents

1	Introduction	8
2	Background	9
2.1	Etherphone	10
2.2	The Modular Integrated Communications Environment (MICE)	11
2.3	Personal Exchange (PX)	13
2.4	IC-Card Telephone System	14
3	Integrating the Workstation and the Network	16
3.1	Intelligent Agents	16
3.2	User Interface	17
3.3	Internetworking	18
3.4	Distributed Call Processing	19
3.5	How is this work different from previous work?	20
4	Implementation	22
4.1	Design Philosophy	22
4.2	Hardware Architecture	23
4.3	Software Architecture	25
5	The Phoneserver	27
5.1	Phoneserver Client Software Library	30
5.2	Network Interface	31
5.3	Client Applications of the Phoneserver	33
5.3.1	Pager	33
5.3.2	QDVM	33
5.3.3	Xphone	34

5.3.4	Forward	34
5.3.5	Logger	35
5.3.6	Activity Server	36
6	Automated Call Management Entity	37
6.1	Call Management	37
6.2	Software Architecture of the ACME	38
6.2.1	Data Structures	38
6.2.2	Socket Manager	40
6.2.3	Call Management Example	42
6.3	Sorting Rules	43
7	The Telephony Language	46
7.1	Challenges of the Telephony Language	46
8	The Phoneditor	49
8.1	Graphical User Interface	49
9	Discussion	53
9.1	Privacy and Security	53
9.2	Difficulties with ISDN	54
9.3	Personal Communications Networks	54
10	Future Work	57
10.1	User Study	57
10.2	Simulations for ACME	58
10.3	Robustness of the ISDN Network Interface	58
10.4	Miscellaneous	58
11	Summary	60
12	Acknowledgements	62
A	Configuring the Phoneserver	63
A.1	Enumerating the Event and Command Structures	64
A.2	Call Appearance States	64
A.3	Interest Structure	65

B	Configuring the ACME	67
B.1	Sample Rule Set File	67
B.2	Configuration File	68
B.3	Alias Files	68
B.4	Protocol between ACME and the Phoneditor	69
C	Lex and Yacc Description of Telephony Language	70
C.1	Lex File - token.l	70
C.2	Yacc File - parser.y	72

List of Figures

1-1	Examples of Call Management	8
2-1	Etherphone Architecture	10
2-2	MICE Hardware Architecture	12
2-3	MICE Software Architecture	12
2-4	PX Hardware Architecture	13
2-5	PX Software Architecture	14
2-6	IC-Card Telephone Set	15
4-1	System Architecture	24
4-2	Software Architecture	26
5-1	Command Structure	29
5-2	Event Structure	29
5-3	IPC Connections	30
5-4	Communicating with the Phoneserver	31
5-5	Call Processing Functions	32
5-6	Pager Program	34
5-7	Forward: Popup Window	35
5-8	Logger Program	35
6-1	Rule Structure	39
6-2	Condition and Action Structures	40
6-3	Socket Manager	41
6-4	Filled Conditions and Action Structures	42
6-5	Unsorted Rule Set	43
6-6	Sorted Rule Set	45

7-1 Intra-Rule Conflicts 47
7-2 Inter-Rule Conflicts: Second Rule Never Reached 48
7-3 Inter-Rule Set Conflicts: Cyclic Transfers 48

8-1 Phoneditor 50

A-1 Call Appearance to Interest Structure 65
A-2 Interest Structures 66

Chapter 1

Introduction

This thesis is an experiment in intelligent call management agents, distributed call processing, user interfaces and internetworking in a heterogeneous computing and telecommunications environment. The call processing services are presented through a graphical user interface and allow the users to customize their communication services to an extent that is not possible in today's telecommunication systems. For example, users can specify to the system statements such as those shown in Figure 1-1.

“No calls between 12 noon to 1pm, except if it's from Peter.”
“Transfer my calls to wherever I am except if I'm in the cafeteria.”
“Transfer my calls to voice mail immediately if I'm not in my office.”

Figure 1-1: Examples of Call Management

The thesis first examines previous related research that has been done at other laboratories in the U.S., Canada and Japan. Chapter 3 discusses the motivation for this work and how it is different from the previous work. Chapter 4 discusses implementation issues of the thesis such as the hardware and software requirements to reproduce this experiment at another lab. Chapter 5 describes the phoneserver in detail - the architecture of the software and the protocol of communications. The automatic call management entity (ACME) used by clients to manage their communications service is discussed in chapter 6. The rule-based scripting language used to instruct ACME is describe in chapter 7 and the graphical interface used for generating these rules is discussed chapter 8. The remaining chapters discuss and summarize the work as well as conjecture as to how this architecture may be used in future telecommunication networks.

Chapter 2

Background

In 1984, the Modified Final Judgment required the Regional Bell Operating Companies (RBOCs) to provide *equal access* to their *inter-lata* exchanges to all long distance competitors. To meet this mandate, the RBOCs rapidly upgraded their old electro-mechanical switches to digital stored program control central office (CO) switches. These new digital switches paved the way for large-scale increases in Centrex services, programmable private branch exchanges (PBX) and feature-rich telephone sets. Such services are now commonplace in the office environment and are beginning to reach the home as well.

Undoubtedly, these new services have added value to the existing telephone systems. However, the added functionality has brought along with it an increase in the complexity of its user interface. Several human factors studies have been conducted in the hopes of providing a user interface that can effectively manage these new services [8, 10, 20, 19]. One approach to this ease-of-use problem is to use the office computer to help manage this complexity. In the past decade, several attempts have been made to integrate the world of telecommunications into the computing environment and to personalize telephony services. Some of the published work outside of the MIT Media Laboratory has come from Xerox PARC, Bellcore, Bell-Northern Research (BNR) and Nippon Telephone and Telegraph (NTT). The Etherphone project, developed at PARC, analyzed the use of Ethernet for providing telephony services [27]. Bellcore developed an intelligent network testbed, called MICE, that was designed to provide an environment for rapid prototyping of customizable telephony features [10]. Unfortunately, both Etherphone and MICE are no longer active projects. At BNR, researchers are working on developing a *Personal eXchange* system to merge voice communications with computer applications [14]. A recent project at NTT has focused on developing a *smart* telephone that would enable the personalization of telephony services [17].

2.1 Etherphone

Etherphone was an experimental telephone system at Xerox PARC that was intimately tied to a computer network. Specialized hardware was used to connect a digital telephone set to an Ethernet local area network (LAN). This hardware enabled the Etherphone to transmit digitized voice, signaling and supervisory information over the network. The system architecture is shown in Figure 2-1.¹ A telephone control server handled routing information and other such responsibilities typical of a PBX. A general purpose computer and a file server were included in the system to provide voice in computing applications. The main engineering challenge was to meet the real-time transport requirements of voice data over Ethernet.

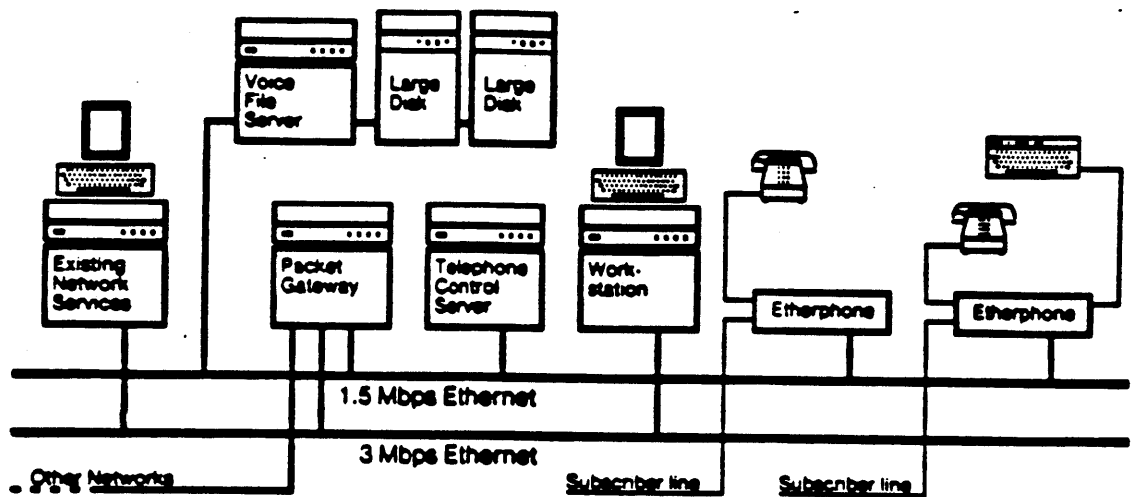


Figure 2-1: Etherphone Architecture

Transporting voice and other multimedia data types along a common bus architecture provides the opportunity for voice call processing and integrating multimedia data into a common computing environment. The ability to perform programmable switching control opens up a whole host of possible applications including directory-based call placement, call logging, call filtering, automatic call forward, voice mail, voice annotation and voice editing.

The cost of this architecture is less reliable telephone service. Telephone service would be severely degraded if the traffic on the Ethernet reached saturation and would fail completely if the Telephone Control Server or the LAN failed. These implications would likely be unacceptable for most companies because telephone service is such an integral part of any organization's operations. Until

¹Figure taken from [27].

computing systems evolve to the stage when their fault tolerance is comparable to today's switches, the integration of telephony and computer system likely will remain impractical. Etherphone was able to overcome this reliability issue by installing a regular subscriber line to each Etherphone. This line was used for telephone access outside the Etherphone system and could be used as a backup whenever the network failed.

2.2 The Modular Integrated Communications Environment (MICE)

The MICE project at Bellcore was an attempt to reduce the complexity of introducing new service specifications in a CO switch by developing a rapid prototyping system for network services.

The authors emphasized three services within MICE – personalization, customization and integration. The idea of personalization meant that services were associated with users and not their telephones. Users could customize their own personal telephone profile through the use of a finite-state table language developed for MICE users. This construct allowed users to use arguments such as the identity of the caller, the time of day and the line status, when making call processing decisions. Thus new services such as *conditional call forwarding*, *variable forwarding*, *routing lists*, and so forth, were made possible. Finally, integration referred to the multimedia integration of data into a common notification, addressing and controlling mechanism. One example of this is the integrated voice and text mailing system, *imail*, that was built for MICE[18]. Imail allowed users to access both electronic mail and voice mail from a computer workstation.

The authors assumed that the computing and telecommunications world of the future would be composed of a heterogeneous network. The components that they used to develop MICE, shown in Figure 2-2, reflected this decision.

The authors wrote the software to interconnect all the components. This task was divided into three modules, as shown in Figure 2-3². The *foundational* software was largely composed of the Central Control Process that acted as a server and communicated with other processes via inter-process communication (IPC) over Ethernet and managed peripherals via a serial link. Resource software provided device drivers and service software was the level presented to users.

The authors later attempted to introduce an ISDN switch to the architecture. However, they ran into more problems than they anticipated. Their hardships were documented in an internal paper [3] and are summarized here. They claim that definitions and extensions of protocols are needed.

²Figures 2-2 and 2-3 taken from [9].

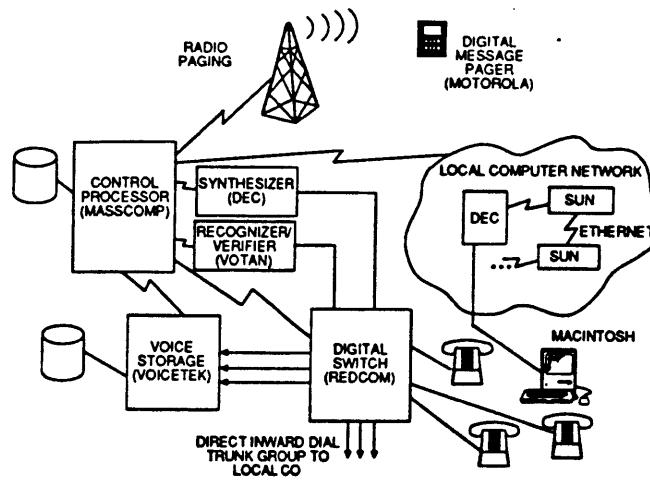


Figure 2-2: MICE Hardware Architecture

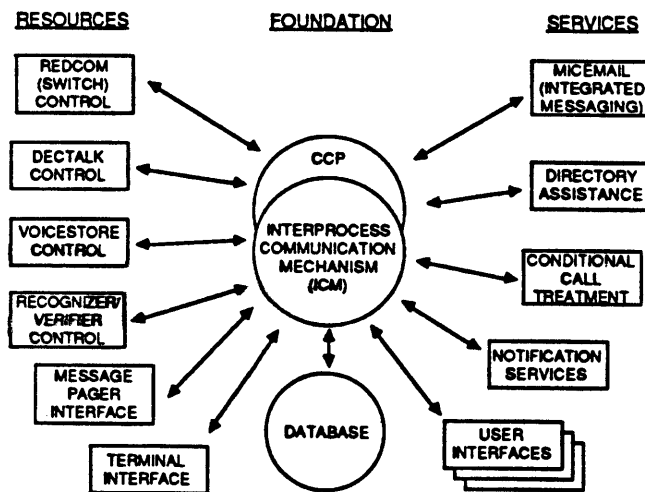


Figure 2-3: MICE Software Architecture

Signaling mismatches exist between the telephony world and the computer communication world. Data compatibility problems exist among different machines. Protocol conformity problems exist between different implementations.

2.3 Personal Exchange (PX)

The PX project is exploring architectures that will enable workstations to conveniently communicate, store, retrieve and process voice as a data type. Researchers at BNR are developing architectures that will enable the workstation to establish voice connections and perform voice switching. By enabling the workstation in this way, they are able to distribute much of the call processing capabilities to the desktop rather than concentrating it at the switch.

The system design of the hardware in PX includes a LAN of workstations where each workstation is associated with a telephone. The workstation is connected to the telephone through an adaptor that also terminates a Northern Telecom ISDN-compatible key system (see Figure 2-4). The key system acts as a circuit server for voice connections and has a 2B+D architecture. The D channel is a message channel that is implemented as a logical messaging bus. By allowing the workstations to read and write messages to this bus through the adaptor, the workstation can instruct the key system to establish voice connections through the B channels. Thus, the key system serves as a circuit switch with an open external control interface.

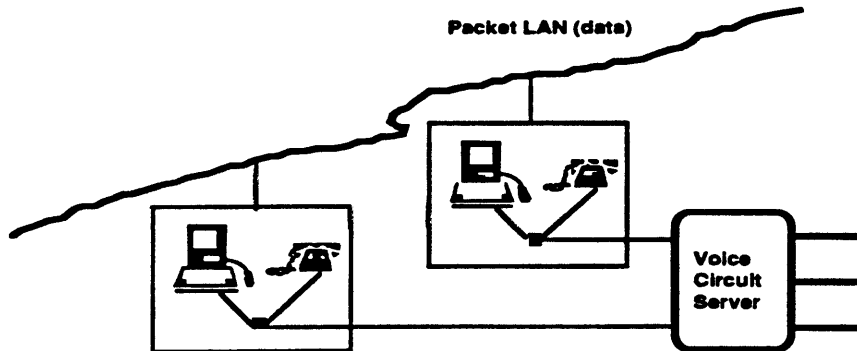


Figure 2-4: PX Hardware Architecture

There are three layers in the software architecture of PX: device drivers, services and applications (see Figure 2-5³). The *telephony management* toolkit in the middle layer is of most

³Figures 2-4 and 2-5 taken from [2].

interest to this thesis. This layer provides high-level abstractions for telephony management services to the application programmer. The abstractions provided allow the programmer to initiate, receive and manipulate calls.

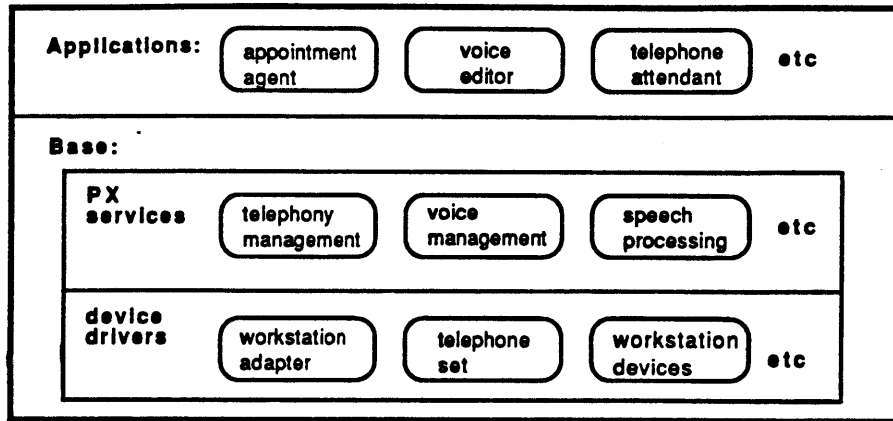


Figure 2-5: PX Software Architecture

The PX project is looking at applications to use the workstation to assist voice communications. to integrate voice and data and to enable delivery and retrieval of data from a workstation through the voice channel. The applications to assist in voice communications are using the workstation to provide a better user interface to advanced telecommunications services. This is done through a screen-based telephony interface and using the workstation as an interactive answer machine. Applications that integrate voice and data include voice annotation of text documents and voice entry to an on-line text calendar. Finally, the workstation is being used to deliver voice messages for applications such as telemarketing and to remotely retrieve electronic mail by reading the text out using a text-to-speech synthesizer.

2.4 IC-Card Telephone System

The IC-Card Telephone System developed at NTT is an attempt to provide personal telephone services. The services that the researchers at NTT wish to provide include: directory dialing, voice calling, automatic transfer, automatic answering and call charge accumulation. They have placed much more hardware in the telephone to enable these services and provide a better interface (see Figure 2-6). In order to provide personalized services, a removable IC-card is used to store information about the user. User information stored on the IC-card such as telephone numbers, dates, times, answering

machine messages, schedules, and so on, are transferred to RAM when the IC-card is inserted into the telephone.

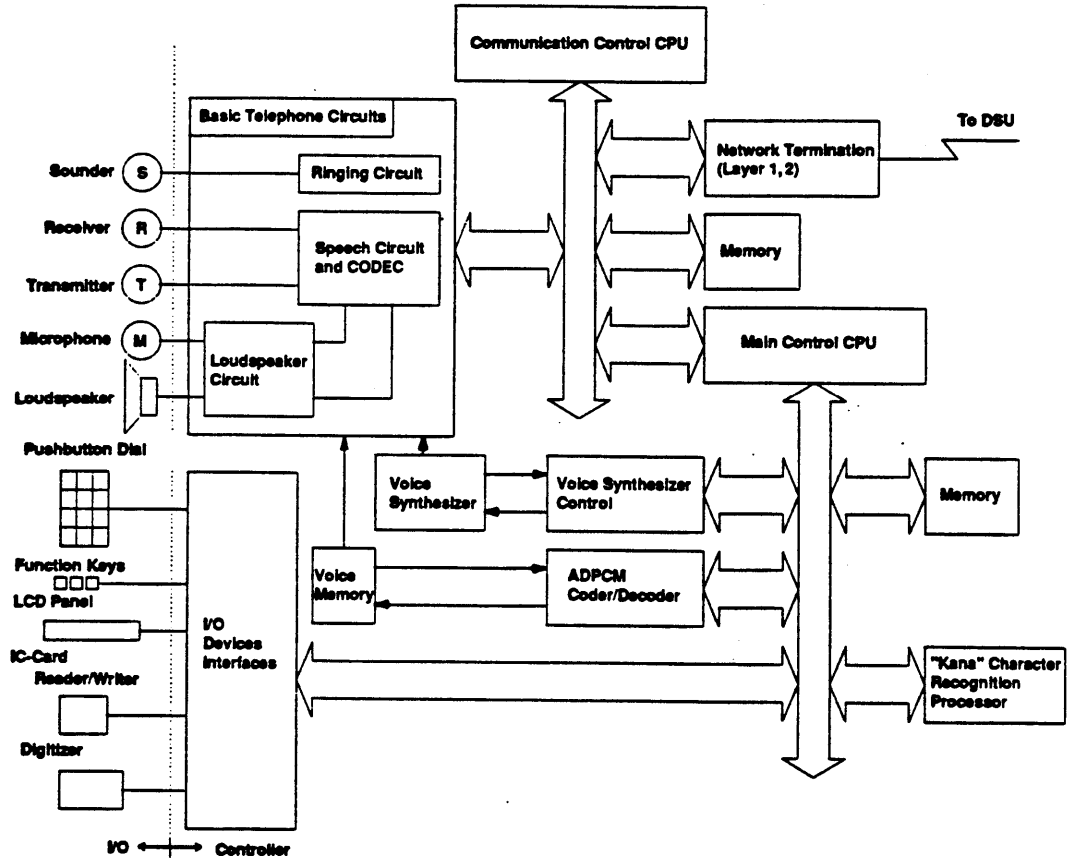


Figure 2-6: IC-Card Telephone Set

Chapter 3

Integrating the Workstation and the Network

This chapter will examine the issues involved in integrating the workstation into the network. In this thesis, the personal engineering workstation will be simply referred to as *the workstation* and the AT&T 5ESS ISDN switch that connects MIT with the rest of the international telecommunications network will be referred to as *the network*. First, the benefits of integrating the workstation into the network will be examined in sections 3.1 and 3.2 and then the architecture needed to provide these benefits will be discussed in sections 3.3 and 3.4. Finally, section 3.5 will examine how the architecture of the system built in this thesis differs from previous related work.

3.1 Intelligent Agents

One way of abstractly viewing the telecommunications industry is that it is a *pointer* industry. The entire network represents a huge storage bank of pointers represented by telephone numbers that people can use to access information or to gain access to other people. Viewed in this way, the telecommunications network facilitates human to human communications by providing links that may span great distances. Unfortunately this communications system sometimes may fail because people are mobile and pointers are static.

Placing a workstation at the periphery of the network as an *end terminal*, gives the network access to the information and capabilities available in the computer world. From a user's perspective, the workstation acts as an agent on behalf of the user to manage the network for him. This may mean processing transactions, scheduling events, making it easier to use this huge network of pointers or

participating in call management by providing the network with the correct pointer.

Consider the list of factors that commonly go into making a call management decision. Clearly, these factors will vary from person to person; in general, however, these factors may be broken down into static and dynamic parameters. Static parameters are those parameters that rarely change with time. For example, a user may always want his telephone to be forwarded to his voice mail when he does not answer his call. Dynamic parameters change with time. An example would be if a user schedules a meeting in her office and wants all her calls to be handled by voice mail for the duration of the meeting. If the workstation can model the activity of the user, it can provide useful and dynamic information to the network about call routing. The dynamic information that can be used at the workstation may include the following: entries from an on-line calendar, the time of day, the location of the user, the status of the user's telephone line, the calling party, a list of people the user has recently called, what activity is currently happening on the user's machine, and so on. The agent built and proposed in this thesis, called ACME, is an attempt at create a dynamic call processing agent.

3.2 User Interface

As mentioned in the previous section, the workstation can be used to make it easier to manage the network. One approach, such as the one described in section 2.4 is to make the telephone more sophisticated. Another approach would be to enhance the switch. The approach proposed in this thesis is to migrate some of the telephony functionality away from the telephone to the computer. It is difficult to see how the telephone can offer user interface comparable to what a workstation can provide, without becoming prohibitively expensive. Also there is motivation to simplify the CO switch as discussed in section 3.4.

While the amount of functionality that is to be moved is unclear, two guidelines can be followed. First, at the very minimum, the telephone set should provide Plain Old Telephone Services (POTS) since POTS is too important to lose when the workstation fails. Thus, the workstation should focus on *value added* network services. Secondly, researchers at Bellcore have found in [10] that the telephone is attractive for non-control functions such as listening to a message or placing a call. However, display devices were found to be more attractive than touchtone keypads for service control of any complexity, such as forwarding or deleting messages.

It is argued in [22] that using display devices, such as workstations, improves the user interface and enhances functionality for several reasons. Workstations have better input devices, such as a mouse and full keyboard, and have superior displays that are larger and easier to read than telephone displays. Also, giving the workstation access to telephony functionality provides an opportunity for

integrating the communications services provided by the network with the communication services available in the computer network, such as electronic mail. Finally, there is more opportunity for personalization of services because the workstation has access to databases.

Recent studies at Bellcore indicate that enhanced network services are hard to remember and often not used [20]. Researchers at Bellcore proposed a mnemonic command syntax for controlling advanced telecommunications services. What they discovered was that the mnemonic command syntax did help users to remember the commands but did not change their pattern of usage. This finding indicates that there exists a significant amount of inertia for introducing any value-added network service. Any new service ought to be useful enough to noticeably add value while being easy enough to use so that the benefits outweigh the cost to the user of having to adapt to the new service. What is needed is a simple set of abstractions that will increase the functionality of the system while reducing the complexity of its usage. The abstractions are mechanisms by which users can conveniently think of and express to the computer how they would like their communication services to be managed. The approach taken in ACME is to use the IF-THEN paradigm to establish service control (see section 6.1) and provide a graphical user interface called Phoneditor, described in chapter 8.

3.3 Internetworking

The intelligent integration of computing technology and the telecommunications world is a subject of international research. Perhaps nowhere is this topic more intensely studied than in the area of ISDN. This should come as no surprise, since integrating a telecommunications network with the digital computer would seem to be a natural evolutionary path for a digital network such as ISDN. In fact, the name ISDN itself, Integrated Services Digital Network, implies the integration of *service* to a digital network.

Internetworking the telecommunications network with the computer network was performed at the network layer as defined by the seven-layered OSI reference model for software. In ISDN, the network layer uses the Q.931 protocol defined by the CCITT. A protocol engine to drive the Q.931 protocol from the workstation end was used (as discussed in section 5.2). The phoneserver resides on top of this protocol engine and communicates with it. Significant time and effort was spent in refining this protocol engine because of the differences between Basic Voice Services defined by the CCITT and Supplementary Voice Services offered by the 5ESS switch running the 5E4 generic program.

Building ACME required both hardware and software integration. The hardware integration involved networking a host of peripherals - ISDN-PC boards, text-to-speech synthesizers, active

badges¹ and workstations. The software integration involved internetworking the telecommunications network with the workstation network and integrating server processes with client and other server processes within the same network. The implementation details are discussed in chapter 4.

3.4 Distributed Call Processing

Distributed call processing is the decentralization of call management decisions from the central office to the end terminal or user. The ACME is distributed in the sense that the intelligence and knowledge needed for call management is handled locally at each workstation, either automatically by computer or manually by human intervention, while all the physical routing remains in the CO.

Distributed call processing is well motivated both from a network and a user concentric view. From a network concentric view, the decentralization of call management decisions from the CO switch can decrease the complexity of the switch. Manufacturers of CO switches are now having to develop and maintain switches that are increasing in size and complexity at a staggering rate. Given this scenario, one approach to managing the complexity is to define a basic set of functions within an open architecture switch and empower customers to easily design and customize their own set of enhanced services. This idea is not revolutionary and is the basic tenet behind the *intelligent network* proposed by Bellcore [7].

In the Intelligent Network/2 (IN/2) proposal for the Public Switched Network in the United States, the network architecture is characterized by distributed call processing and modular “building blocks” of services called Functional Components (FCs). The intent is to meet the telecommunication and information needs of customers by allowing them to arrange their network services on demand. The current service introduction cycle is two to five years for a new service to go from concept to production [6]. Using IN/2, customers can create their own services by interfacing to the switching system. The FCs, which previously resided in the switch, reside in an intelligent peripheral outside the switch. Signaling information is passed between the customer premise equipment, the switch and the intelligent peripheral using the Signaling System 7 (SS7) protocol. Customers build services by combining FCs which can be thought of as reusable modular capabilities.

This concept of decentralizing functionality is seen in the computing world as well. Consider the evolution of computer operating systems. The growing size and complexity of the operating system led to the notion of placing only what is essential to a computer system in the *kernel* of the operating system and providing access to the kernel through an applications interface or system calls.

Decentralization of call management decisions also makes intuitive sense from a user services

¹ This is a recent technology for tracking location of badge wearers.

perspective. Consider the list of static and dynamic factors that commonly go into making a call management decision. A feature-rich CO switch can be programmed (through a 12-button telephone keypad plus a few function keys) to hold most of the static parameters people commonly use in call management decisions. The AT&T 5ESS switch has access to the configuration table which, for a given telephone line, provides static information on what to do if a call is not answered or if the line of the called party is busy, and so on. However, it would be difficult if not impossible for a CO switch to access and maintain dynamic information on all the users that it serves except for the time of day and the telephone number of the calling and called party. It is much more likely and feasible that a personal workstation would have access to dynamic information about the user. Hence, distributed call processing allows the system to decentralize the decision making to the more knowledgeable workstation.

3.5 How is this work different from previous work?

ACME differs from the Etherphone project because in ACME the voice is circuit switched in the central office instead of carried over Ethernet. Also, in Etherphone, all the call processing is performed at the centralized *telephone control server* which keeps track of state and sets up all connections. The IC-Card telephone system is also different in the philosophy of its architecture. This approach is building more functionality into the telephone set. The emphasis in this thesis is in building network based telephony services with distributed call processing capabilities. Value is added to telecommunications services by migrating some of the functionality from the telephone set to the workstation. This chapter has argued that this approach will be more beneficial than enhancing the telephone set.

The MICE and PX projects have the most similarity to ACME. The main difference in architecture between the MICE project and ACME is that in MICE, all call management is performed from a centralized process called **Central Control Process (CCP)**; whereas in ACME each user has an ACME process managing calls on their behalf. In MICE, the CCP is passed a static database in the form of a finite state table and responds to phone events based on this static table. By providing an ACME process for each user, the system can respond to dynamic information about a user and account for those factors in call processing (see section 3.4). The PX project also has an architecture of distributed call processing and more call processing is actually performed at the personal workstation in PX than in ACME. The **Configuration** window [2] used in PX is similar to ACME; however, PX does not use it for dynamic call routing. There is more emphasis placed on managing voice in a personal workstation than in using the workstation for call routing.

None of the projects discussed in this chapter attempt to model the user dynamically in order to make call management decisions. The use of BRI is another difference. PX is using an ISDN-compatible key system and MICE tried to integrate ISDN into the system. Of the difficulties encountered in MICE, some similar difficulties were encountered in the ACME project. In particular, different implementations of ISDN made it hard to port ISDN network handlers from one site to another. The differences in the protocols caused difficulties that are documented in section 5.2.

Chapter 4

Implementation

4.1 Design Philosophy

The emphasis in the ACME project is to build a distributed system. All the hardware is distributed over a LAN. The workstations can be of different architectures although currently only Sun workstations are being used. Indeed, the phoneserver which was originally developed on a Sun 2 workstation using a Motorola 68020 processor, has been ported to a Sun 386i workstation using an Intel 80386 processor, and then was ported to a Sun Sparcstation 1 using a Sparc processor. The software that is used in ACME is distributed over several processes and processors.

There is one centralized process for *controlling* the call processing functionality. This process is the phoneserver. The ACME processes that implement the *decision making* are distributed. Each user runs an ACME process to act as an agent on their behalf at their local workstation. The motivation for distributing the decision making processes is to allow the ACME processes to collect and use information about a user dynamically in call management. This is much more effectively done in a decentralized architecture than in a centralized one. Another advantage of distributing the system is that it provides more fault tolerance. By having different servers run on different machines, the whole system does not stop if one machine fails. Only the resources provided by the server on the machine that failed are no longer available. In fact, separate processes residing on the same machine are sufficiently isolated so that even if one process malfunctions it will not degrade the services of other processes.

Since the ACME system is distributed across several hardware platforms and many software processes, the integration of these subsystems becomes very important. These systems are integrated through a protocol that is designed to be architecture independent. The processes communicate via

inter-process communication. Indeed, this whole integration process is part of a bigger picture within the Speech Group of integrating many *desktop audio* tools in the workstation [21].

Another requirement of the design is that all the servers should run asynchronously. The software is designed to act as an interrupt handler to external stimuli. This is implemented in various forms using *callback* facilities provided in the X Athena Widget set and the *Socket Manager* (see chapter 6.2.2) and the `select` system call in Unix. It is important that the server processes be asynchronously driven because the servers must be able to receive events while handling events and cannot guarantee a real-time response. Fortunately, when dealing with interfacing to human response times, no stringent response times are placed on the phoneserver. Perhaps the only situation where a response is absolutely required within a finite time is when the 5ESS polls a stimulus set to see whether it is "alive". Responding to this message is appropriately handled at a layer below the phoneserver – either in the network handler process in the Sparcstation 1 or in the Teleos board on the PC. This real-time requirement is shielded from the phoneserver.

4.2 Hardware Architecture

The MIT campus installed an AT&T 5ESS switch in August 1988. The switch, running the 5E4 generic, provides ISDN Basic Rate Interface (BRI) to the entire campus.¹ The D-channel (or the data channel) provides end-to-end digital communication via a 2B+D time division multiplexed architecture. The D-channel is a 16 Kbps out-of-band signaling channel. The concept of *common channel signaling* (CCS) is likely to become a foundation for telecommunication services in the future. CCS is a great improvement over previous in-band signaling switches because it provides greater flexibility in receiving and processing information. The two B-channels (or bearer channels) are 64 Kbps bit streams. Voice is carried by standard pulse code modulation while data information is typically transported by a protocol imposed by the subscriber. The BRI lines are terminated at the customer premise by an S interface which typically plugs into an AT&T ISDN 7506 telephone set. 7506 sets are uniquely identified by one primary directory number (PDN) call appearance and physically have room for nine more, not necessarily unique, secondary directory number (SDN) call appearances. However, the actual limitation of the 5ESS switch on a BRI line is 1 PDN and up to 63 SDNs. Running a BRI line with 64 call appearances to the phoneserver can monitor up to 64 telephones by having access to their D-channels.

Figure 4-1 shows a BRI terminating in a workstation; however, when the ACME project first

¹ISDN is a prohibitively large topic to address adequately in this thesis. The interested reader is referred to reference [23].

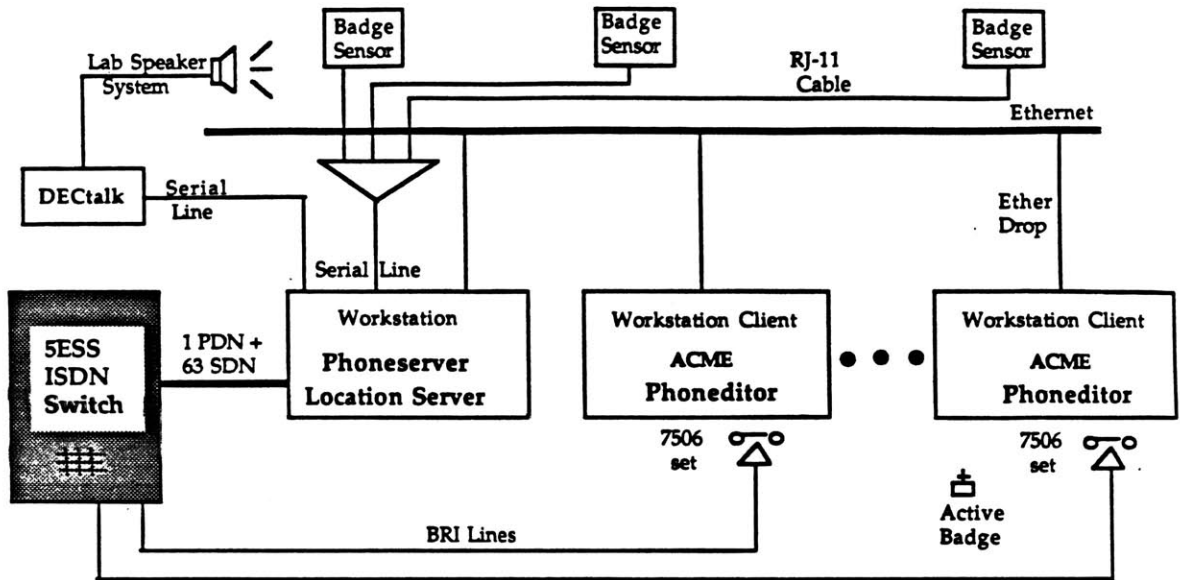


Figure 4-1: System Architecture

began, a Teleos B100PC card that resided on an IBM PC bus was used as an interface to the 5ESS because no similar ISDN board was commercially available on a Unix platform. The PC was used as a slave to the phoneserver process residing in the Sun workstation. Logically, the PC and the server on the workstation were a single entity. The PC delivered asynchronous messages to the phoneserver whenever an event occurred. The information transmitted was identical to the information contained in the event structure described in chapter 5 except that the last three fields in that structure were not transmitted. The *calling_name* and *dn* fields were filled by the phoneserver which had access to calling table information. The *time_of_day* was filled using Unix time.

As the project evolved, an attempt was made to eliminate the PC interface to the 5ESS completely. A Sparcstation 1 with a customized motherboard and BRI plug-in line was obtained, along with proprietary ISDN device driver code from a Speech Group sponsor. These two ISDN interfaces are further discussed in section 5.2.

Active badges are a new technology from Cambridge-Olivetti Computer Research Laboratories. The badges are roughly 2.25 inches by 2.25 inches by 0.25 inches and weigh a few ounces. They can be comfortably worn like any company identification badge. Each badge emits a unique infrared signal every 15 seconds. The signals are received by sensors that in turn send a message via RJ11 phone

cables to a concentrator that sends a message to the *location server* process through a serial line. This server process constantly monitors the location of each person wearing a badge and can be queried for information by client processes [26].

The DECTalk text-to-speech synthesizer is used to provide paging. Incoming calls are announced through the internal speaker system of the Media Lab (see section 5.3.1).

4.3 Software Architecture

The Teleos B100PC board comes with a device driver. The device driver handles all the messaging with the 5ESS up to and including layer 3 – Q.931. This is a convenient level of abstraction for the application programmer. The PC is essentially acting as a parser and a gateway to the 5ESS. The software extracts relevant signaling information from the switch such as calling line identification, time stamps and ISDN calling, to pass **onto** the phoneserver residing in the Sun workstation.

All services operate on a client-server based paradigm. The phoneserver communicates with the clients via unit sockets. A client side software library provides applications with an interface to the phoneserver. The phoneserver provides the basic telephony services to a client at a programming level. The programming interface is described in section 5.1.

It is assumed that the client processes with a graphical interface will be running within the X Window system and have access to graphical tools such as icons and pop-up notification windows. The client processes have an automated call management entity (described in chapter 6) interfacing with the phoneserver and location server on its behalf. The interface to the phoneserver is done through the client software library.

The interaction between processes is illustrated in Figure 4-2. The circles denote processes while the square denotes a database. The figure shows only one ACME process; however, each client has at least one ACME process to interface with the phoneserver. The protocol between the ACME and the phoneserver is architecture-independent, allowing clients to run on different machines with different processors.

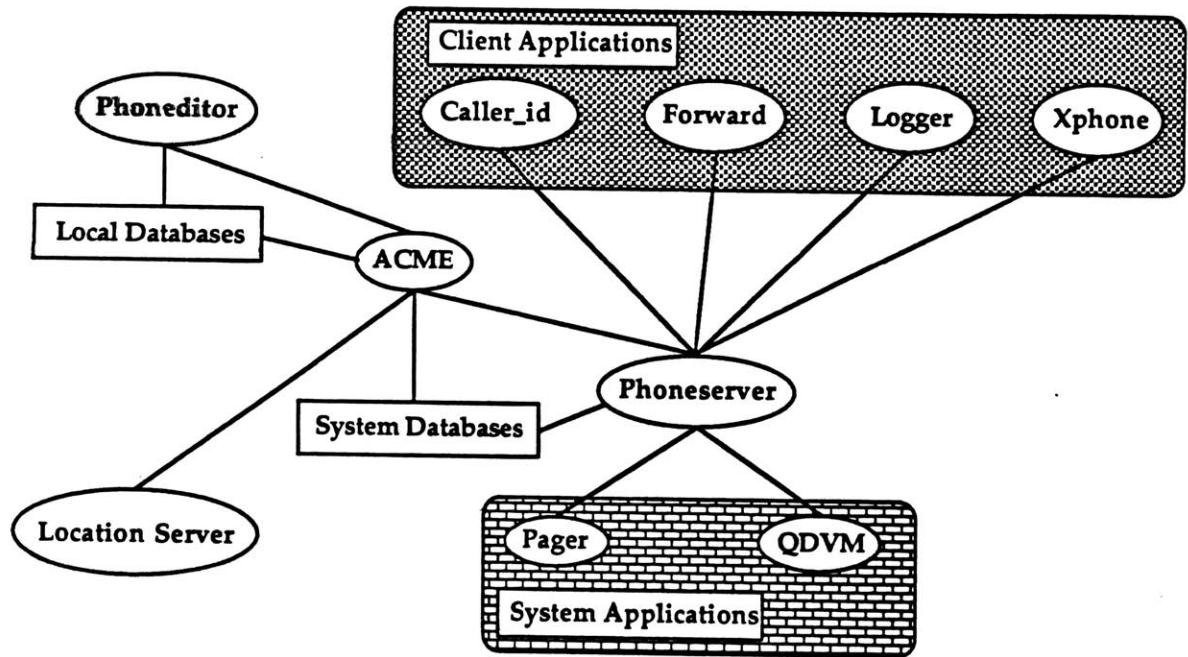


Figure 4-2: Software Architecture

Chapter 5

The Phoneserver

The phoneserver is an asynchronous process that provides telephony-based network services to clients. It receives its input from three sources: new clients registering for services, existing clients sending it messages and signaling information from a BRI line.

When the phoneserver is first started, it loads a calling table and a translation table into memory and initializes some states for each *call appearance* (CA). The calling table maps a *directory number* (DN) ¹ into a name in order to personalize the services that are offered. Another table is used to translate a set of CAs into DNs where each DN may have one or more call appearances that appear on one or more telephone sets. The phoneserver uses a BRI line that has 64 call appearances of 35 different DNs that are of interest to this project. The owners of most of these DNs work in the Terminal Garden of the Media Lab and may be interested in using the network phone services the Speech Group is building. Finally, the phoneserver initializes the state of each CA to be IDLE in an internal array at startup time. Clearly, this may be a false assumption to make about all 64 call appearances, especially if the phoneserver is started during office hours. However, there is no known way of easily obtaining the status of a call appearance from the switch and little damage is done if a wrong assumption is made since it will correct itself after the first event occurs on that call appearance.

The phoneserver communicates with clients through socket-based interprocess communications. The Internet domain format is used to establish sockets between the phoneserver and its clients. The Internet domain is the Unix implementation of the IP/TCP/UDP suite of protocols from the DARPA Internet standard. Addresses in the Internet domain format are composed of a machine network address plus a unique number referred to as a *port*. The protocol allows for communications between processes in the same machine and between processes residing on separate machines that are of the

¹Commonly known as a phone number.

same or different architecture. The messages are exchanged through a *stream* socket. The actual socket is created using the Unix *socket* system call. Stream communications is a connection-oriented circuit that provides reliable and error-free data communications. No message boundaries are imposed and **the network will** manage all the problems of fragmentation, ordering and error correction to ensure the integrity of the message that is passed. **After the socket name** has been bound to the socket using the **bind** command, reading and writing from the stream is accomplished by using the system calls **read** and **write**, respectively. This newly created socket will be referred to as the *phoneserver socket*.

After the phoneserver socket has been created, the phoneserver listens to it to see if any clients are trying to connect to it. On the client side, a similar sequence is followed to create a stream-based socket. This socket is then connected to the publicly known port number and host machine by the client using the system call **connect**. This action signals the phoneserver to accept the first connection on the queue of pending connections and creates a new socket with the same properties of the *phoneserver socket*. The Unix operating system then assigns a new file descriptor for the newly created socket. A phoneserver client software library is provided to hide this level of detail from the application developer and present the appropriate level of abstraction. Functions to connect to the phoneserver, along with other functions, will be discussed later in this chapter.

Information from a BRI line is obtained from either the Teleos B100PC card or directly from the Sparcstation ISDN device driver (see section 5.2). The phoneserver supports both sources in the same way. If the B100PC card is being used, a serial port is opened and its file descriptor is stored. If the Sparcstation ISDN device driver is used, a socket is opened with a network handler process called *nhatt* (see section 5.2) and the socket number is stored.

At this point, the phoneserver has three sets of peers to communicate with – new clients connecting through the phoneserver socket, a list of sockets from existing clients and either a file descriptor for the serial port or a socket number for *nhatt*. The phoneserver must listen to all of these sources and serve them as the need arises. The system call **select** elegantly fills this need. Before calling **select**, the socket numbers and file descriptors of every peer is masked into one master file descriptor called *masterfds*. A copy of *masterfds* called *readfds* is passed to the **select** call along with the timeout parameters, which are set to block indefinitely until a message arrives. When a message **does** arrive, the **select** statement sets the *readfds* mask to the file descriptor or socket number that received the input. The application may then fetch the message by issuing a **read** command on *readfds* and appropriate action may then be taken on the message.

If the input was from a new client, then the connection request is accepted, the client is added to a linked list of existing clients and the *masterfds* is updated to include the new client. If the input is from an existing client than there are two possibilities. Either the client is closing the connection (or

has died) or is sending a command. If the client is closing the connection, the phoneserver receives a zero length message and proceeds to close the socket, free the information block about the client from the linked list of existing clients and update the masterfds by unmasking the closed socket. If the message is a command, a the structure shown in Figure 5-1 is received and is passed to a parser that determines the command type, extracts information from the relevant fields and executes the appropriate command. The command messages are sent as large byte-oriented character strings to provide architecture independence.

```

struct command {
    char command[CMD_SIZE]; /* type of command */
    char ca[MAX_CA_LEN]; /* call appearance */
    char src[MAX_SRC_LEN]; /* originating source */
    char dest[MAX_DEST_LEN]; /* destination */
    int interest; /* clients interest */
}

```

Figure 5-1: Command Structure

Depending on the type of command, a message may be returned to the client. In general, acknowledgements between the source and the receiver are not used because reliable data communications is assumed to be provided by the stream-based socket IPC as described above. If the message was from nhatt or the B100PC card, an event structure shown in Figure 5-2 is filled.

```

struct event {
    char state[STATE_LEN];
    char ca[MAX_CA_LEN];
    char ISDN_call_info[ISDN_CALL_INFO_LEN];
    char calling_dn[MAX_DN_LEN];
    char forwarded_from_dn[MAX_DN_LEN];
    char calling_name[MAX_NAME_LEN];
    long time_of_day;
    char dn[LOCAL_DN_LEN];
}

```

Figure 5-2: Event Structure

Upon receiving an event packet, the phoneserver looks up the information block corresponding to the call appearance, which is identified by the field *ca* . If the owner is interested in the event that occurred as indicated by the field *state*, then the entire data structure is sent to this client. The

internal organization of representing the interest of the clients is described in Appendix A.3.

5.1 Phoneserver Client Software Library

The phoneserver client software library provides a C programming interface to the phoneserver.

The functions `init_phoneserver_ipc` and `reconnect_pserver` (shown in Figure 5-3) open connections to the phoneserver and return a socket. `init_phoneserver_ipc` initializes the connection to the phoneserver. `reconnect_pserver` is used when the phoneserver fails and a reconnection is required. It first closes the existing socket, then calls `init_phoneserver_ipc` every *sleep_count* seconds until a connection is finally made. It is important to “sleep” between attempts to reconnect to the phoneserver; otherwise, the port used by the phoneserver socket will be constantly busy and the phoneserver never will be given the chance to bind to the port. Sleeping for 60 seconds between reconnection attempts has been found empirically to be sufficient amount of time for the phoneserver to bind to the port and for most phoneserver client applications to sleep. This technique for reconnecting to the phoneserver is inadequate for other asynchronous servers connecting to the phoneserver that cannot afford to go to sleep. One solution to this problem is to use a *timeout callback* construct provided by the *Socket Manager* (see section 6.2.2). This construct allows the program to execute `init_phoneserver_ipc` periodically without blocking at that point in the program.

```
int init_phoneserver_ipc()

int reconnect_pserver(socket, sleep_count)
    int socket;
    int sleep_count;
```

Figure 5-3: IPC Connections

After opening a socket with the phoneserver, a client is known to the phoneserver. However, the phoneserver still does not know what type of services the client wants. The function `register_interest`, shown in Figure 5-4, allows the client to tell the phoneserver what its interests are. The *dn* parameter specifies what DN the client is interested in. The *interest* parameter can be one of 13 defined states plus a `WILD_STATE` (see Appendix A.2). The `WILD_STATE` registers an interest in every event that occurs on the specified DN. The most common states of interest are idle, active, incoming and dialing. To register an interest in more than one state but not all states, the defined states can be masked into one integer and passed as the *interest* parameter. `Register_interest` will

return a **SUCCESS** or **FAIL** to the client. Failures usually occur when the client registers an interest in a DN not known to the phoneserver.

After the client registers an interest in particular events, the phoneserver will send the event structures described in Figure 5-2 to the client. The function **check4input** checks if any messages have arrived in the socket specified by the parameter *device* . The *blocking_time* parameter specifies the length of time in seconds to block on the device. A value of -1 will block indefinitely. When an arrival occurs, a non-negative value is returned and the client can retrieve the message by calling **get_event** and passing it a pointer to an empty event structure.

```
int register_interest(dn, interest, sock)
    int dn;
    int interest;
    int sock;

int check4input(device, blocking_time)
    int device;
    int blocking_time;

int get_event(inbuff, sock)
    event inbuff;
    int sock;
```

Figure 5-4: Communicating with the Phoneserver

The remaining functions shown in Figure 5-5 are self-explanatory except for two cases. The **phone_status** function queries the phoneserver as to the state of a particular DN of interest and returns the state. The *relative_ca* parameter in the **call** function allows the programmer to select which call appearance to use on a particular telephone set. It is relative because this number is to be translated to an absolute call appearance on the **BRI** line used by the phoneserver.

5.2 Network Interface

As discussed in section 4.2 the ACME project evolved from terminating a **BRI** line at an IBM PC to a Sparcstation 1. The Teleos B100PC comes with a device driver and application software library called ASK100. A ***VOICE** library within ASK100 provides an **interface** to Supplementary Voice Service messages. It is interrupt driven and applications can send and receive ISDN signaling information through **NetBIOS** [25]. Upon receiving an event the application will load the first five fields of the data

<pre> int call(src, dest, relative_ca, sock) char *src; char *dest; char *relative_ca; int sock; </pre>	<pre> int hold(ca, sock) char *ca; int sock; </pre>
<pre> int call_pickup(ca, sock) char *ca; int sock; </pre>	<pre> int phone_status(dn, sock) char *dn; int sock; </pre>
<pre> int conference(ca, dest, sock) char *ca; char *dest; int sock; </pre>	<pre> int transfer(ca, src, dest, sock) char *ca; char *src; char *dest; int sock; </pre>
<pre> int drop(ca, sock) char *ca; int sock; </pre>	<pre> int unhold(ca, src, sock) char *ca; char *src; int sock; </pre>
<pre> int forward(src, dest, sock) char *src; char *dest; int sock; </pre>	

Figure 5-5: Call Processing Functions

structure shown in figure 5-2 and send it to the phoneserver through a serial connection.

A proprietary ISDN network handler, *nhatt*, for the Sparcstation 1 provided by a Speech Group sponsor is being used. Nhatt is a Unix process that receives D-channel information from the memory resident ISDN device driver. A finite state table describing the Q.931 protocol is used as a protocol engine to communicate with the 5ESS through the D-channel. The nhatt process connects to client applications via inter-process communications. The phoneserver is a client and receives asynchronous packets of ISDN signaling information from the nhatt process. Again, like in the case of the Teleos card, this packet is received and used to fill the data structure shown in figure 5-2.

5.3 Client Applications of the Phoneserver

A host of client applications were written to utilize the phoneserver. This section will discuss two system applications – pager and qdvm – and three individual client applications - xphone, forward and logger. The section will also examine the relationship between the phoneserver and another server called the *activity server*.

5.3.1 Pager

A paging program was written to serve the phoneserver clients working in the Terminal Garden. A public *called file* contains a table of strings to be announced when an incoming call arrives for a given DN and a *calling file* contains a table of names associated with particular DNs. For the sake of privacy, users can choose to not have the identity of the calling party announced or not have the call announced at all.

Pager is very popular with the students that work in the Garden. The presence of music in the Garden often drowns out the sound of ringing phone in nearby offices. Pager helps to alleviate this problem for those people who have offices close enough to the Garden to respond to the announcement.

The program itself is very straightforward. Pager first loads the calling tables into memory. Then it opens a socket and registers an interest in the incoming calls of all DNs known to the phoneserver. Finally, it sits in a loop, waiting for events and announcing the appropriate string through the Garden speaker system using a Dectalk text-to-speech synthesizer.

5.3.2 QDVM

QDVM is a voice mail system running on the Speech Group Sun workstations and provides voice mail services for the entire Speech Group [13]. Electronic mail is sent to the voice mail recipient to inform him of the voice mail. There is a graphical user interface and a telephone interface to the voice mail.

```

get_tables(calling_table, called_table);
sock = init_phoneserver_ipc();
register_interest(WILD_DN, REG_incoming ,sock);
while(1) {
    get_event (allocated_event_structure, sock);
    output_to_dectalk (allocated_event_structure, calling_table, called_table);
}

```

Figure 5-6: Pager Program

All calls to the Speech Group telephones are forwarded to a central number used by QDVM. QDVM uses the phoneserver to find information about the calling party and the called party.

5.3.3 Xphone

Xphone provides a variety of automated telephone dialing service from a workstation. It was originally reported by the paper [22] as "Phonetool". Xphone uses a combination of mouse and keyboard entry to place and receive calls. A "Rolotool" application was written to provide an on-line rolodex that works in concert with Xphone. The full functionality and advantages of such an interface are reported in [22].

Xphone was recently ported to use the ISDN phoneserver. Xphone uses the calling processing functions `call` and `drop` from the phoneserver client software library shown in Figure 5-5 to place calls and drop calls. One current limitation inherent in the 5ESS switching system is that there is no way to place an AT&T 7506 telephone set on *speaker phone* mode through software control. Thus the user is forced to either pick up the handset or hit the *speaker phone* button on the 7506 set to receive calls.

5.3.4 Forward

One phenomenon that was observed with the introduction of the pager was that people began to run to their offices to catch an announced call. A mechanism to transfer the call to the Garden phones would be very useful. One way to implement this idea is for a client application to register an interest in the incoming calls of a particular DN. When an incoming call occurs, the student may be notified at her workstation and given the option to transfer the call to a phone through her workstation.

The *forward* application was written for this task in the X Window Systems[11]. When an incoming call occurs, the workstation "beep" is sounded and a popup window appears on the screen (see Figure 5-7). The user is told that he has an incoming call from a given person (or number, if one is known) and given four pushbutton commands to select from. He may choose to transfer the call to the phone nearest his workstation, transfer the call to a secretary, transfer the call to voice mail or

dismiss the call completely.

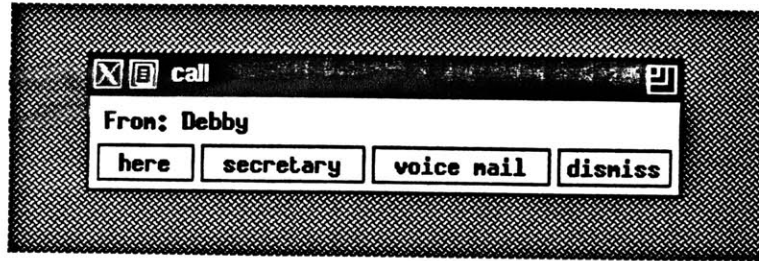


Figure 5-7: Forward: Popup Window

This program was functional for a short time, but was not reliable enough to offer to people outside the speech group. The transferring function was not robust enough to withstand sustained usage.

However, a successful application that evolved from the forward application is *caller_id*. *caller_id* is the forward application with no selection buttons. Displaying the call on the screen is much more readable than on the 48 character LCD display on the AT&T ISDN 7506 set. This feature a much more effective way of alerting the user to the identity of the calling party before picking up the receiver.

5.3.5 Logger

The *logger* program was written originally to debug the phoneserver and was later enhanced to retrieve data for the study proposed in section 10.1 Though the study was not conducted in this project, *logger* proved to be one of the most useful utilities from the user concentric view. The most interesting event in everyday use is the incoming call. Thus, the *logger* opens a socket with the phoneserver, registers an interest in REG_incoming on the telephone number of the user and then sits in a loop waiting for events from the phoneserver and recording the events (see Figure 5-8).

```
sock = init_phoneserver_ipc();
register_interest(my_dn, REG_incoming ,sock);
while(1) {
    get_event (allocated_event_structure, sock);
    record (allocated_event_structure, output_filename);
}
```

Figure 5-8: Logger Program

An interesting side effect of this utility is its social implications. It is considered to be courteous

to return a telephone call. The people who are aware of the fact that I am logging calls may expect me to return a call even if they did not leave me a voice mail message. However, unlike voice mail, when the logger or phoneserver processes fail, there is no way for the calling party to discern that the call is not being logged. This lack of feedback may cause users of the logger program to unknowingly violate social expectations when the logging process fails.

5.3.6 Activity Server

The activity server provides high-level information about the current activity of a set of users [16]. It receives information from three sources: the finger server, the location server and the phoneserver. Based on this information, the activity server makes inferences about the activity of a user at a given time by establishing interdependencies and resolving conflicts in the information retrieved. It may be used as a tool within a trusting user community to coordinate office activities.

The activity server requests state information from the phoneserver. The critical state information required by the activity server are: onhook, offhook and incoming call states.

Chapter 6

Automated Call Management Entity

The Automated Call Management Entity (ACME) is a process that is associated with a single DN. Its function is to manage the telephony services as specified by a user for a particular DN. To accomplish this task, the ACME interprets a rule-base created by the user and interfaces with the phoneserver to manage the events that the phoneserver reports.

6.1 Call Management

Consider how one manages a telephone call. Decisions as to what to do with an incoming call are made based on many factors, including: the calling party, the time of day and the activity that one is currently involved in.

The first two items, calling party and time, are straightforward to obtain from the network. The Q.931 protocol delivers to the called party information fields that contain the calling party's number and time of day. A much more difficult problem for a workstation is to determine the current activity of a person.¹ The information that is available to workstations in the Media Lab comes from various sources. The location of active badge bearers on the third floor of the Media Lab is available to the ACME from the location server. The current phone status for clients of the phoneserver is available to the ACME from the phoneserver. Finally, information about users is available on-line from the user's calendar and from the operating system.

Making inferences about a user's activity based on calendar information is a very complex

¹This problem is addressed in Sanjay Manandhar's Masters thesis [15].

problem, unless the calendar entries are extremely structured and simplifying assumptions are made about each entry. One solution may be to format the calendar so that each line begins with a block of time, where a filled block implies that the user is busy. However, the granularity of the information provided in this solution is not detailed enough to be useful to an ACME. Also, Speech Group members use the Unix calendar file format with an X Window interface, *Xcal*, that enables the inclusion of sounds within the calendar. The presence of sound files and the fact that the Unix calendar file format permits unconstrained text entry into each date makes the problem of deducing the activity of a user at a given time to be difficult.

Information provided by the Unix operating system and its associated software tools does provide a tractable solution to obtaining information about the activity of the user. The `finger` command in the Unix operating system can provide information as to where a user is logged in and how long the user has been idle. This information is available to the ACME from the *finger server* [15]. In this era of highly networked workstations that permit remote logins to various host machines, the combination of both login information and idle time is necessary to reliably determine at which workstation a user is physically working. The finger server finds the hosts with the least idle time and traces back hosts until a host console is found. The workstation with the host console is assumed to be where the user is physically working [15].

6.2 Software Architecture of the ACME

6.2.1 Data Structures

The ACME is an asynchronously driven event manager. It is a standalone process that sets up its initial conditions by reading ASCII database files containing aliases and rules for call management. It receives external input events from two sources - the phoneserver and the phoneditor. External inputs are managed by a *Socket Manager* [12] that provides socket-based inter-process communication. After the external inputs are received, they are tested against the database of conditions and appropriate actions are taken.

The user can set up a personal alias file to supplement the system-wide alias files. These aliases are used for convenience and also manifest themselves in the *options* button within the dialog box of the Phoneditor (see Figure 8-1). All personal alias files found in the root directory of a user supersede system-wide aliases. The most common alias files used are `.cme_rooms` and `.cme_group_names`, which specify aliases for a group of locations and a group of people respectively (see Appendix B.1 for examples). The ACME reads these alias files and maintains a linked list for each alias. The ACME manages one other linked list, containing telephone numbers of people that the user recently called.

This list is used in the situation where the calling party argument is specified to be *someone I recently called*.

A user's rule set is also placed in the user's root directory, in the file `.cme_rules.<DN>`, where `<DN>` is an **extension** representing the DN for that rule set. At execution time the ACME reads this file, parses it, sorts (see section 6.3) and loads the rules into three data structures – a linked list of rules, a linked list for the conditions in each rule and a structure for the action to be taken. The rule structure is shown in Figure 6-1. The *num_conditions* field contains the number of conditions for a given rule while the *rule* field contains the actual rule string. The *condition_ptr* and *action_ptr* are pointer to the condition and action structures (as shown in Figure 6-2), respectively. Finally, the *next* field is a pointer used to link the rules together.

```
struct rules {
    int num_conditions;
    char rule[MAX_RULE_LEN];
    condition *condition_ptr;
    action *action_ptr;
    struct rules *next;
}
```

Figure 6-1: Rule Structure

Each rule has one or more instances of conditional structures and exactly one instance of an action structure. The condition structure contains four fields and a pointer to allow for a linked list. All conditions in the linked list are combined by the boolean operator *and*. The *type* field contains information on the type of condition: calling party, time, location, workstation or telephone status. The *operator* field contains a logical operator such as *not*, *before*, *after* or *around*, that is to be applied to an operand field. The two operand fields, *op1* and *op2*, contain information entered by the user at the time the rule was created such as names, telephone numbers, room numbers, time, machine names, and so forth.

The action structure contains three fields and is considered only when all the conditions for that action have been fulfilled. The *type* field contains information on the function to trigger while the *op1* field is a parameter passed to the function. The operand may contain information such as names or telephone numbers. The *delay* field specifies the number of seconds to wait before triggering the action routine.

```

struct condition {
    int type;
    int operator;
    char op1[64];
    char op2[64];
    struct condition *next;
}

struct action {
    int type;
    int delay;
    char op1[64];
}

```

Figure 6-2: Condition and Action Structures

6.2.2 Socket Manager

At this point ACME has initialized all its internal data structures and is ready to communicate with other processes. It is a client to the location server ² and phoneserver and a server to the phoneditor. Thus, it establishes connections with the location server and the phoneserver and then adds a new network service called *acme_port* for the Phoneditor to connect to. This is all done through the Socket Manager. The Socket Manager provides management of sockets between clients and their corresponding server and is typified by its use of asynchronous callback routines. It is loosely modeled on parts of the design of the X Window System.

The mechanism by which sockets are managed in ACME is described in Figure 6-3. First, **SmInit** is called to initialize the Socket Manager. **SmOpen** connects the process to the server identified by the first argument and returns the socket number. **SmAddService** adds a new service to the server and the service name is translated into a port number for clients to bind to. When clients bind to this port or communicate with it, the **SmNewSocketCallback** routine will invoke the **newclientcb** routine to execute. The first **SmSetReadCallback** routine watches for data in *sock_phone* and calls **phone_handler** when some input arrives and also passes the pointer *phone_data*. Similarly, this is done for the location server. Finally, **SmMainLoop** is called after all the callbacks are set up. It dispatches events to the callbacks when events occur and never returns.

One call missing from Figure 6-3 is ACME registering an interest with the phoneserver in incoming and outgoing call events for a specified DN before **SmSetReadCallback** is issued. The ACME could be coded to register an interest in more types of events; however, incoming and outgoing call events are the events that have been found to be useful thus far. Since the ACME registers an interest in only one DN, it can **manage** calls for only one DN. If a user has more than one DN to manage,

²ACME ideally should connect to the activity server for more reliable data. However, at the time of writing the ACME code, the activity server was not reliable enough for use since it was also under development. See section 10.4 for further discussion.


```

SmInit();
sock_phone = SmOpen("phoneserver", "");
sock_location = SmOpen("location-server", "");
SmAddService("acme-port");
SmNewSocketCallBack(cme-port, newclientcb, NULL);
SmSetReadCallBack(sock_phone, phone_handler, phone_data);
SmSetReadCallBack(sock_location, location_handler, location_data);
SmMainLoop();

```

Figure 6-3: Socket Manager

separate ACME processes have to be started for each DN. There is a line-level option to specify the DN to monitor. Associated with each DN is a rule base and a port number. The rule file for each DN has a specified filename ending with the extension of the DN and the port number is found by table lookup in a configuration file. Currently, three service ports within a Speech Group LAN have been dedicated for IPC between the phoneditor and the ACME. This number can be increased simply by dedicating more ports.

External inputs from the phoneditor are in the form of commands. The commands are **update**, **sleep** and **wake**. The inputs will cause **newclientcb** callback routine to execute and parse the command. The **update** will cause the ACME to purge an old rule set and read in a new way. The phoneditor can request the ACME to go to sleep (stop managing events from the phoneserver), wake up or update its rule base after the phoneditor has modified it. For all commands, the ACME will either acknowledge the successful completion of a command with an **ACK** message or send back a **NAK** or negative acknowledgement to inform the user that the command failed. These commands are further discussed in section 8.1 and Appendix B.4.

External inputs from the phoneserver are in the form of events. The inputs will cause **phone_handler** to execute. **Phone_handler** retrieves the event packet and matches it to the condition structure. If all the conditions are fulfilled, then an action is fired after the delay specified in the *delay* field of the action structure. The delay mechanism is implemented by the timeout routine, **SmSetTimeoutCallBack**, provided by the Socket Manager. The simple **sleep** command available in Unix is inadequate for implementing the delay because as in any asynchronous event manager, the ACME must monitor incoming events even during the delay. In addition to monitoring, the ACME must also keep the state of **all** call appearances because **some** action functions are contingent on the status of the DN.

6.2.3 Call Management Example

The Terminal Garden is a large computing facility where many students in the Media Lab work and socialize. The paging facility announces incoming calls over the Garden speaker system. It is a common occurrence that students have to run back to their offices to grab the incoming telephone call before it is transferred to voice mail. Thus a useful rule for the ACME would be to increase the alerting time before a call is transferred to voice mail if a student is near the phone but not in her office. For example:

“IF I am in the Garden and the calling party is important-person THEN transfer the call to voice mail after 15 seconds.”

For this example, ACME’s parser will fill the condition and action data structures as shown in Figure 6-4.

	<u>Condition1</u>	<u>Condition2</u>		<u>Action</u>
type:	LOCATION	CALLING_PARTY	type:	TRANSFER
operator:	Nil	Nil	delay:	15
op1:	me	important-person	op1:	vmail
op2:	Garden	Nil		

Figure 6-4: Filled Conditions and Action Structures

When an incoming call from an *important-person* arrives for the user and he is in the Garden, the status of his phone is recorded to be INCOMING and then the `SmSetTimeoutCallback` routine is set to call the `transfer` routine after 15 seconds. The argument *important-person* is an alias set by the user in the `.cme_group_names` file and is represented internally as a linked list that is passed to the `phone_handler` for matching. The ACME continues to monitor incoming phoneserver events as normal. After 15 seconds have elapsed, the `transfer` function is called and the status of the phone is checked. If the status is still INCOMING, then the transfer to voice mail proceeds. If the status has changed. (i.e., the user has picked up the phone and the status has become ACTIVE) then the action is killed. Upon exiting from `transfer`, the `SmSetTimeoutCallback` is reset so as to not execute anymore. A current limitation of the Socket Manager is that only one `SmSetTimeoutCallback` can be outstanding. Thus, if an event occurs between setting the timeout and the callback routine being executed, then that event must wait until the previous callback is finished.

If an action is successfully fired, the `phone_handler` returns control to `SmMainLoop`; otherwise, it will try to fire another rule. The action may fail for various reasons. One reason may be the case stated above where the action is no longer valid after a change of state. Another reason may be that

the ACME rejects an invalid action such as transferring a call to *where I am* where the *where I am* parameter is either an unknown location or the same phone (see Figure 7-1).

6.3 Sorting Rules

If only one rule could be true in a rule set at a given time, no sorting would be necessary. But, for any given event it is possible that none or all of the rules can be true. In the example shown in Figure 6-5, if my friend Bill calls after 6 p.m. and I am working in the Garden on the machine shasta, then all the rules are true except for the last one. Which action should the ACME fire? A trivial solution is for the ACME to follow exactly what the user specifies and fire the first rule that tests to be true. The intuitive answer seems to be that the most *important* rule should be fired. However, using this approach, the ACME must assume that its metrics for measuring importance reflect the way the user thinks: otherwise, the ACME appears unpredictable. A fundamental tradeoff is made here between the desire for predictability of the ACME and the autonomy given to the ACME to make inferences.

begin

IF the calling party is friends THEN transfer the call to where I am after 0 seconds.

IF the calling party is Bill and the location of me is not my office THEN transfer the call to my secretary after 3 seconds.

IF the location of me is Garden THEN transfer the call to voice mail after 15 seconds.

IF the status of the machine shasta is logged in THEN notify me by a pop-up window.

IF the call party is anyone and the time is after 18:00 THEN transfer my call to voice mail after 0 seconds.

IF the status of my phone is active THEN do nothing.

Figure 6-5: Unsorted Rule Set

It can be argued that the ACME should be given more autonomy to make inferences rather than be constrained to behave predictably for the following reasons. First, the ACME is a distinct agent from the previous efforts in call management because it is trying to use dynamic information in its call processing. The use of dynamic information argues for ACME being intelligent and flexible and not constrained to a static script. On the other hand, it is important that ACME behaves predictably to the user; otherwise, the user will lose confidence in it. This reasoning argues that the ACME should carry out the rule-base in precisely the way that the user has written it. However, having the ACME obey rule-base exactly does not guarantee predictability. Humans are not as logical as computers and tend to forget mundane facts, such as rules, much faster than computers. Thus, even if ACME is behaving in a manner that is logical and consistent with its instructions, the user may not see it this

way because she misunderstands the logic of her rule set or has forgotten some rules.

The basic assumption that is made is that the *precision* of a rule is equated to its *importance*. The more important rules are tested first; therefore, after the rule-base is loaded the rules are sorted from the most important to the least important.

First, the rules are sorted by the number of conditions within a rule. Rules with more conditions are assumed to be more specific than rules with fewer conditions. Before the sorting algorithm is performed, all rules are scanned for the word “anyone”. Rules containing this word in the condition are assumed to have one fewer condition than there actually is, since the “anyone” condition is always true. A rule containing one condition and the keyword “anyone” will be sorted to be the last rule and will *act as the default* condition since it will always fire. Thus generating a rule like, “If the calling party is anyone then transfer the call to voice mail after 3 seconds,” means that the call will go to voice mail if nothing else happens.

Next, rules with the same number of conditions are sorted according to the highest priority condition type within a rule. The order of priority from most important to least important is as follows: phone status, calling party, location, time and workstation status. The phone status condition is most important because if a user is already in a phone conversation then he is physically at his telephone set and his decision about how to manage the call should override any ACME action. Furthermore, when a user is active, it restricts the phoneserver ACME from routing calls anyway because there are not enough CAs (see 7.1). Thus, a useful rule for all users to have is the last rule in Figure 6-5. The second most important condition is the calling party because that gives specific arguments and in general the identity of *the calling* party is very important to call management. The location condition can also be precise; however, as a factor in call management it is intuitively less important than the identity of the calling party. Perhaps this is because there are only two common scenarios with respect to location – either a person is near her office phone or she is not. If the person is away from her office, then the desired call management behavior generally will not change as a function of location. However, for the calling party condition, there may exist several people or several sets of people for which the desired call management behavior differs. The time condition specifies a range in which the condition is true and thus it is not as precise as the previous conditions. Finally, the workstation condition is thought to be the least important because there is the weakest association between that and telephone service. Its utility is probably limited to aiding in locating a person and alerting him.

After sorting between conditions, like conditions are sorted by argument. Tokens are judged to be more specific than aliases which are composed of a list of tokens.

The unsorted rule set in Figure 6-5 is sorted in Figure 6-6. In this rule set *friends* is an alias which contains Bill as a token. Clearly, the sorted rule set that is represented internally to the ACME

differs from the unsorted rule set which is saved to the rules file `.cme_rules.<DN>`. This may lead to some uncertainty. To aid the user, it would be useful to display how the ACME is internally representing the rule set. This topic is discussed in section 8.1.

`begin`

`IF the status of my phone is active THEN do nothing.`

`IF the calling party is Bill and the location of me is not my office THEN transfer the call to my secretary after 3 seconds.`

`IF the calling party is friends THEN transfer the call to where I am after 0 seconds.`

`IF the location of me is Garden THEN transfer the call to voice mail after 15 seconds.`

`IF the calling party is anyone and the time is after 18:00 THEN transfer my call to voice mail after 0 seconds.`

`IF the status of the machine shasta is logged in THEN notify me by a pop-up window.`

Figure 6-6: Sorted Rule Set

Chapter 7

The Telephony Language

The telephony language developed for ACME is a very simple English-like language, based on the IF-THEN construct, for call management. The possible conditions are the parameters that are available to the ACME to model user activity, as discussed in section 3.4. They include the identity of the calling party, user's locations, the time and the status of a workstation or telephone. The possible actions are currently limited to transferring a call to a different location and alerting users their workstation and through the paging system. The list of possible actions can be extended to include all the actions available to the phoneserver as the need arises.

The language is implemented using the lex and yacc Unix programming tools. An abbreviated version of the program listing illustrating the grammar in a pseudo-Backus-Naur Form is displayed in Appendix C. The lex and yacc files generate C functions are used by ACME. These functions read the rules files, parse the rules and fill the rule, condition and action structures shown in Figure 6-1 and Figure 6-2.

Users can specify more than one condition per rule. All the conditions in a rule must be satisfied before the action will be executed or *fired*. There is one rule set per DN and the users can specify an unlimited number of rules per rule set. Users can specify the rules in any order they like; however, the order of specification is significant since it affects the behavior of the ACME.

7.1 Challenges of the Telephony Language

Specifying a rule set for call management is a challenging problem because conflicts can arise for several reasons. There are timing problems, intra-rule conflicts, inter-rule conflicts and inter-rule set conflicts.

Timing problems arise from the fact that there is a delay between when a condition is tested as being true and when an action fires. Within this delay, the state of the phoneserver may change and

subsequently nullify an action. Consider the last example in Figure 7-2. If Peter calls, the ACME will test the condition to be try and fire a **transfer** in four seconds. However, if after one second I answer the call, the ACME can no longer transfer the call (nor do I want it to!) to voice mail.

Timing problems also arise due to race conditions. If more than one ACME process manages a single DN, then this may lead to events where two ACMEs are competing for the attention of the phoneserver.

Intra-rule conflicts arise because the telephony language is implemented as a context-free language without constraints. Thus, the language specification will allow for *over production*. In other words, it is possible to generate well-formed rules that have nonsense or illogical semantic meaning. For example, Figure 7-1 displays two rules that have correct syntax but are flawed. In the first example, if I am in my office when an incoming call arrives, then the phoneserver will attempt to transfer the call back to my own phone. This procedure will fail on the standard BRI line with three CAs because there will be insufficient CAs for the phoneserver to use. In the second example, the action will never fire since both conditions cannot be simultaneously true.

“IF the location of me is my office THEN transfer to where I am after 3 seconds.”

or

“IF the calling party is Phil and the calling party is Marilyn THEN transfer the call to voice mail after 5 seconds.”

Figure 7-1: Intra-Rule Conflicts

Resolving conditions between rules will help to prevent situations where certain rules are never reached. For example, in Figure 7-2, the latter rules in both examples are never reached since the ACME processes test the rules sequentially. It is possible to resolve this rule set and eliminate the rule deemed to have a lower priority. However, if the rule with the higher priority is placed before the rule with lower priority, then the correct rule will fire and no inter-rule resolving is necessary. The ACME assumes that the more recently created rule has a higher priority. Thus, a newly created rule is placed at the front of the set of rules.

Resolving rules across rule sets in a distributed environment is an even tougher problem. Consider the example shown in Figure 7-3 of two separate ACMEs running on two separate rule sets that have an inter-rule set conflict. In this example, if either party receives a call, the calls will be transferred back and forth in an endless cycle.

It will be more important to deal with these *negative interactions* [1] as more advanced conditions and actions are added to the ACME. For example, call-waiting and call-forward-on-busy are

“IF the time is before 18:00 THEN notify me by a pop-up window.”
 “IF the time is before 17:00 THEN transfer my call to voice mail after 3 seconds.”
 or
 “IF the calling party is Peter THEN transfer my call to where I am after 0 seconds.”
 “IF the calling party is Peter THEN transfer my call to 3-0673 after 4 seconds.”

Figure 7-2: Inter-Rule Conflicts: Second Rule Never Reached

<p>“IF the calling party is anyone THEN transfer my call to Peter after 0 seconds <u>Chi’s Rule File</u></p>	<p>“IF the calling party is anyone THEN transfer my call to Chi after 0 seconds” <u>Peter’s Rule File</u></p>
--	---

Figure 7-3: Inter-Rule Set Conflicts: Cyclic Transfers

incompatible features for the same call appearance. Another example of a negative interaction, shown in Figure 7-3, is if user Chi forwards calls to Peter who himself has calls forwarded to Chi.

Fortunately, none of the problems described above is known to crash the system even if they are allowed to occur. In the case of cyclic transfers, the calling party will probably hang up after waiting a short time without an answer. In the case of cyclic forwards, the 5ESS will terminate a call that hops between telephones after a few cycles and give the calling party a fast busy signal. However, though these will not crash the system, they should be addressed because they can lead to unpredictable and confusing results for the user. Rather than using formal mechanisms in the language to alleviate these problems, policies in the Phoneditor and the ACME are used to prevent or resolve the conflicts.

The timing problem, addressed in section 6.2.2, is resolved by using callback routines. The race conditions can be prevented if each user is disciplined but only managing their own DN. A less trusting solution is proposed in section 10.4. The intra-rule conflicts are either prevented by the Phoneditor as discussed in section 8.1 or resolved in the ACME as discussed in section 6.3. The inter-rule conflicts are alleviated by the sorting and execution method employed by the ACME as described in section 6.3. No solution has been attempted or found for the inter-rule set conflicts. This is a problem inherent to distributed databases and may be resolved only at a central location such as the phoneserver. Such a solution is beyond the scope of this thesis.

Chapter 8

The Phoneditor

The Phoneditor is a graphical user interface to call management. It is written in the C language using the Athena Widget set of the X Window system under X11 R4. The output of the Phoneditor is an ASCII test file called `.cme_rules.<DN>` (see Appendix B.1). This file acts as a script for the ACME to follow.

8.1 Graphical User Interface

When the Phoneditor process is running, a small icon (displayed in the top part of Figure 8-1) appears on the screen. The icon is a picture beckoning the user to specify how to manage calls. Clicking on this icon will popup the Phoneditor window displayed in Figure 8-1. Clicking on the icon while the Phoneditor window is already displayed will popdown the window.

The layout of the Phoneditor is as follows. The top line of the window is an "instruction" line that provides contextual instructions to the user. Instructions at each point in the interaction are provided to guide the user. The large icon selection box immediately below the instruction line contains all the "condition" and "action" icons. All the condition icons are displayed on the left-hand side of the screen except for the "My Phone" condition icon in the center. The "ACME Status" label above the "My Phone" icon is used to inform the user as to whether the ACME process is actively managing calls on his behalf. The arrows flowing into and out of the "My Phone" icon convey the idea of managing incoming calls and indicate that the natural flow of the screen is from left to right. The condition icons are placed on the left side to denote the initial conditions for an incoming call. The "action" icons are the right to denote what actions to take after an incoming call arrives.

Condition icons must be selected before any action icon may be selected. Upon clicking on a condition icon, a condition is generated and displayed in the "new rule" line that is discussed below.

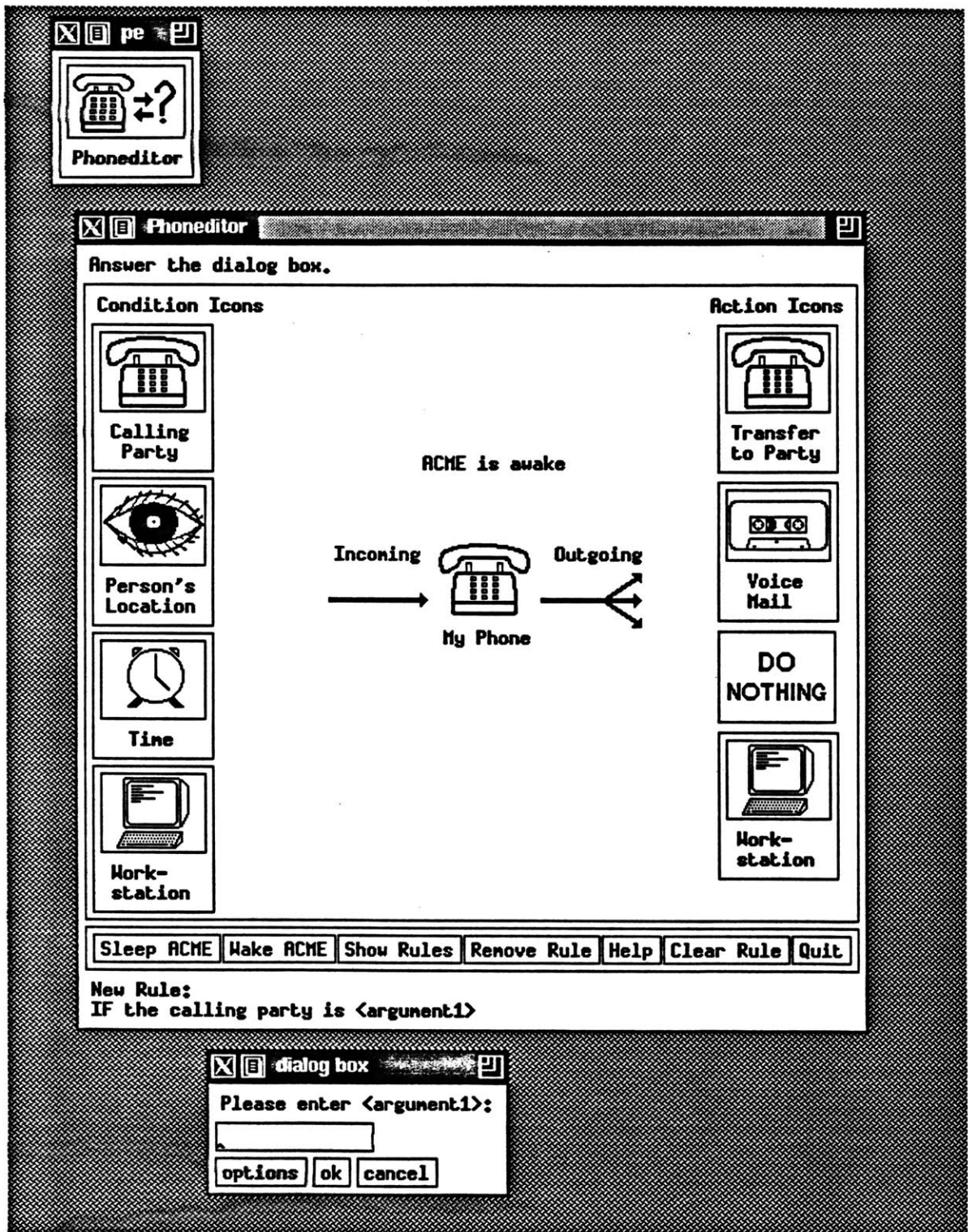


Figure 8-1: Phoneditor

Also, a popup dialog box polls the user to enter how ever many arguments are required for the condition. The user may enter the argument by typing it into the dialog box or by using a pulldown menu available under the *options* button in the dialog box, as shown in Figure 8-1. This button is provided only if a menu is available. Users can set up their own options menu by creating the alias files described in section 4.3. Selecting more than one condition will cause all conditions to be combined by the boolean *and* operator¹. The “My Phone”, “Calling Party” and “Location” condition icons cannot be selected more than once to prevent intra-rule conflicts discussed in section 7.1. The “Time” icon can be selected exactly twice to allow users to create upper and lower bounds on time. There is no limit on the number of times the “Workstation” icon can be selected. After completing one or more conditions, an action icon may be selected. The same interaction that is used to complete a condition icon is used to complete an action icon. Only one action icon can be selected and upon completion of a selection, the rule is written to the `.cme_rules.<DN>` file and an **update** message is immediately sent to ACME. The ACME purges the current rule set in its memory, reads in the new rule set and acknowledges to the Phoneditor that an update has been made. This constant update may seem very inefficient; however, the protocol is designed to ensure that the rule set contained in the file `.cme_rules.<dn>` is consistent with the rule set that the ACME is currently using internally.

Below the icon selection box is a row of buttons. The Phoneditor interfaces to the ACME through the Socket Manager. There may be times when users wish to manage their telephone by themselves without having to kill the ACME process. The *Sleep ACME* button will signal the ACME to stop processing all calls and the “ACME Status” label will be updated to inform the user that the ACME is asleep. The converse of this action is performed by hitting the *Wake ACME* button. The *Show Rules* button is provided to allow the user to see the sorted rule set that the ACME is using. Selecting the button will display in a popup window the rule set contained in the file `.cme_rules.<dn>`, which, as discussed above, is necessarily consistent with the rule set that the ACME is currently using internally. This facility helps make ACME more predictable. To remove a rule, the user can double click on the rule and then hit the *Remove Rules* button. Hitting the *Hclp* button at any time will popup a window of context sensitive instructions for how to proceed. The *Clear* button is used to clear a rule that is in the process of being created. Finally, the *Quit* button is used to exit the Phoneditor application.

Below the set of buttons is the “new rule” line. This line displays the rule as it is being created. In Figure 8-1, the user has began a new rule by clicking on the “Calling Party” condition icon and is being asked to enter who the calling party is in *argument1* . The new rule line displays the contents

¹The boolean *or* operator is not supported because the rules are tested sequentially. Therefore all rules with the same action are combined together effectively by an *or* operator

currently in the rule.

Chapter 9

Discussion

9.1 Privacy and Security

The issues of security and privacy have been ignored in this thesis. It is not difficult to imagine that this type of software can be abused. The main issues of concern regarding privacy are caller identification and electronic surveillance. The issue of providing caller identification is currently being debated in the telecommunications world and is more of an issue for the SS7 protocol than for this thesis. Electronic surveillance of telephone activity can easily be accomplished by using the logger program, discussed in section 5.3.5, which logs all the activity of a DN.

The main concern for security is telephone or PBX fraud. The open interface to the phoneserver makes it easy to re-route calls through the `transfer` routine. Thus it is possible to *steal* incoming calls meant for a called party. It is also possible for a user to dial into a phoneserver line and instruct the phoneserver to transfer the call to a long distance number and thus avoid paying long distance charges.

The ACME project has avoided these issues for two reasons. First ACME is meant to operate within a trusted community of users to provide value-added services to their communication needs. Thus, an attitude of trust prevails. Secondly, the main focus of the ACME project is on functionality. In order to implement a truly secure solution, an authentication server such as Kerberos, is required to authenticate the identity of each client process. This added complexity and substantial investment of time in learning network security is beyond the scope of this thesis.

9.2 Difficulties with ISDN

The difficulties encountered with interfacing the Sparcstation 1 with an ISDN BRI line have been encountered previously in the MICE project. The researchers working on MICE found protocol conformity problems exist between different implementations [3]. The ISDN network handler, nhatt, was originally developed on an ISDN PBX provisioned to offer Basic Voice Services. The AT&T 5ESS ISDN switch supplies Supplementary Voice Services. There are two noticeable differences between the protocol for Basic Voice and Supplementary Voice Services. Supplementary Voice Services has an extra **Associated Type** message used to inform members of a Key System group of the status of a call. Also, Supplementary Voice uses a **Locking Shift Codeset 6** in the **Call Setup** message which contains a few extra fields. These additions were put into nhatt.

An annoying feature discovered in building ACME is that the Q.931 protocol does not allow a functional terminal to transfer a call without first answering it. The functional terminal must receive the call, issue a **transfer** message to the switch, place a call on a free call appearance and then issue another **transfer** message. Thus, calling parties who are dialing long distance are billed for calls as soon as the ACME transfers it even though the call may not be successfully completed.

9.3 Personal Communications Networks

The introduction and interest in widespread tetherless portable radio communications has led to a surge of articles in *Personal Communications Networks* (PCN) in the communications journals in the last few years. It is generally recognized that the deployment of tetherless radio communications will take two evolution paths -- high-powered vehicular cellular mobile systems and low-powered handheld portable sets. Both markets are experiencing tremendous growth. The cellular mobile system in the U.S. alone had two hundred million customers in 1988 [4]. By early next century, cars with factory-equipped cellular phones could easily increase the number of mobile cellular phone users to 100 million [5]. The tremendously successful introduction of cordless telephones indicates that the demand for low-powered handheld portable sets is also as strong. Cordless telephones were introduced in the U.S. in the late 1970s and sales grew to two million units by 1982. Since 1982 roughly four to six million units were sold each year [4]. With such a rapid deployment of tetherless portable radio communications, it's reasonable to assume that portable handheld telephone sets will become ubiquitous in the urban centers of the U.S. in the not too distant future. If this assumption is true, does it imply that wireline-based call management tools, such as the ones proposed in this thesis, will become obsolete? The answer to this question is, "No", for two fundamental reasons -- the nature of the PCN network and the utility of the ACME within this network.

The vision of a Personal Communications Networks in the U.S. is rapidly evolving, although it is lagging behind that of the Europeans vision who have already deployed PCNs in a small scale in the form of the British CT-2 effort [24]. The view put forward by Bellcore is that personal communications should enable a person to initiate or receive a call from anywhere within regions of reasonable population densities [5]. For economic, political and technical reasons there seems to be no easy migration path from the cellular mobile system to the handheld portable sets. For pragmatic reasons, handheld sets must be light and pocket-size. This requirement means that smaller batteries must be used and that high-power electronics are precluded, to increase the mean time between battery rechargings and also for safety reasons. Whereas high-powered vehicular cellular mobile systems operate in the range of one to ten watts ¹ and cover an area of greater than three kilometers, low-powered handheld portable sets must operate in the range of 0.001 to 0.01 watt range and cover an area of less than 400 meters. The shorter coverage area means that the cell sizes would be smaller and as a result the number of radio access ports for a given area would increase. Thus for economic reasons the PCN network would have to be introduced first to areas of high population densities such as factories, apartments, airports, shopping malls, and so forth. For economic and political reasons as well, these radio access ports will be integrated into the existing local exchange networks. Since the switch networks are already in place, there is no need to duplicate the effort. Also, using the existing wireline infrastructure would impose a standard and provide access to a universal network – both of which are needed to make PCN a truly ubiquitous service.

From a technological standpoint, using the local exchange network also makes sense because PCN would be able to piggyback on the intelligent network services that would be provided. Since the PCN network will be coupled to the local exchange network, one or more processes performing similar functions to the ACME will be useful in locating users, filtering calls and routing calls. The fact that the PCN *microcells* are so small means that the network must locate the user to determine which radio access port to service. One mechanism proposed for tracking a handheld set is for the set to contain the user's identity in memory (or in a "smart card" inserted by the user) and have an internal emitting device transmit this unique identity number [5]. The nearest radio access port that receives and decodes this number signal reports it to a central process interfacing with the local exchange network to direct traffic. This scenario is analogous to the setup of the location server. However, using the ACME to inform the network of the location of a user may be more reliable because it employs more sources of information than just badge location to determine the actual location of a user. Also if the user is moving, the network must be instructed to perform *handoff* from one microcell to another. This is *analogous to call transfer* in the local exchange network. In this scenario, the alerting mechanism

¹It is safe for car phones to operate at this power because the antenna is placed outside of the car.

that is performed out-of-band may alert the handheld set and the ACME simultaneously. The ACME may then tell the intelligent network which microcell to interface with.

These call routing and resource arbitration issues will become even more important if the plan to implement a *personal number* calling service is adopted. Many major telecommunications companies are working on a service plan to assign one number to a person and place the burden of alerting the user on the network regardless of how many kinds of telephones he has at home, at work or in the car. The ACME would be used here to inform the network where to route the calls.

Filtering calls will be useful regardless of whether a communications network is wireline or wireless based. For the busy office worker, it will be necessary to filter out the unimportant calls or be able to log calls while he is busy. This filtering process will become even more important if a plan for assigning a personal number is actually adopted. Given this service, it would be important to be able to divide personal calls from business-related calls so that users can handle all their business calls while at work and let the ACME handle the rest. While this is currently beyond the capabilities of the ACME, some technology should be provided to aid the called party and protect her privacy since the network has made it easier than ever before to reach her.

The utility of ACME in the wireline network can be seen as providing greater mobility and a level of call filtering. The ACME in the wireline network provides greater mobility because it uses more *channels* of alerting mechanisms. Users can now be alerted by phones other than their own, by a pager and by a workstation. The mobility is not quite as great as a tetherless system, since after being alerted, the user still must be close to a telephone. Call filtering is performed by having the workstation manage some calls without having to alert or interrupt the user.

For the reasons stated in this section, there will still be a need for software to perform functions similar to those performed by the ACME in an environment of widespread tetherless radio communications. The ACME will not be rendered useless but be used to augment and enhance a network such as PCN.

Chapter 10

Future Work

10.1 User Study

In many user interfaces it is difficult to measure the success of an interface because it is difficult to find an objective metric of measurement. However in ACME there are some objective metrics that could be used. Using the logger program described in section 5.3.5, it is possible to measure telephone activity before and after the installation of the ACME. Some metrics that can be objectively measured are listed below.

- Does this service increase the call completion percentage?
- How many calls went to voice mail and of those calls how many people left a voice mail message?
- Does this service increase the time spent on the phone?
- Do people who use this service use more telephony features such as transfer, forward, hold, etc. than before? Can they remember the services better than before?

The use of the ACME raises some other questions of interest that are more appropriately addressed in the form of the user study. These more subjective questions are listed below.

- In what way does the service replace or augment a human secretary?
- How does this service compare with having a cellular telephone?
- How does this service affect how people view their telephone services?
- How is system usage affected by system reliability?

10.2 Simulations for ACME

To aid in making the ACME more predictable, it would be useful to be able to simulate different types of incoming calls and their associated conditions. Perhaps a companion to the Phoneditor could *graphically generate* event packets that would send phoneserver to the ACME under real conditions. The output of such an event could be visually displayed. An alternative to manually generating these packets is to run the log file that is generated by the logger program through the simulator and display how real world telephone activities would have been handled for a user.

10.3 Robustness of the ISDN Network Interface

For ACME to become a practical working system the code for the ISDN network handler, nhatt, must become more robust. At the crux of automated call management is the **transfer** function. This routine is not working for practical purposes. Once nhatt has been fully debugged, the ACME can reach its full potential and start experimenting with other call processing routines such as `call_pickup` for voice messaging systems. The forward program, described in section 5.3.4 can be deployed on a larger scale. Perhaps this is a good first client application for people to introduce them to call management from a workstation.

10.4 Miscellaneous

Databases are used by the phoneserver, Phoneditor and ACME. These databases are used to convert CAs into DNs, DNs into paging strings, alias names into lists, room numbers into telephone numbers, names into numbers and vice versa. Currently the databases are distributed over several locations. Each server process has users in its own database. However, there is considerable overlap in information between the various databases used by the different servers. In the interest of database consistency, this information should be stored at one location.

ACME is currently connecting to the location server for information on the location of a user of interest because at the present time the activity server is not robust enough for use. As was argued in this thesis, it would be much more reliable to connect to the activity server for this purpose since it uses many more sources of information. Also as the activity server expands to incorporate even more information, such as the user's calendar file, ACME will be able to take advantage of this information as well.

Race conditions in the phoneserver may arise if two separate ACMEs manage one DN. One solution is for the phoneserver to implement a first come first serve policy and refuse to service the

latter ACME. However, there are situations when it is useful for an ACME to monitor the events of another DN though not necessarily manage it. For example, a user may be more inclined to receive an incoming call if he knows that his secretary is already busy on the phone. Thus the phoneserver may offer **two different** classes of registering interest in a DN – managing and monitoring. In both classes, clients receive the event packets of interest but only in the managing class is a client allowed to manipulate telephony services of a DN. To prevent race conditions, the phoneserver simply has to restrict the number of managing classes to one per DN.

Chapter 11

Summary

A system was built to *enable* distributed call processing by internetworking in a heterogeneous computing and telecommunications environment. By integrating the workstation in telephony, a better user interface to advanced telecommunications services was provided by providing *value added* network services that focused on service control. By distributing call processing, intelligent call management agents that could model the user could participate dynamically in call processing.

A phoneserver was developed to enable the platform described above. It bridged client applications in the computing world to the call processing services available to an ISDN basic rate interface. A *client* software library was developed to interface client applications to the phoneserver through inter-process communications. All applications developed by the Speech Group used a client-server model and ran asynchronously.

The main application developed in this thesis was an automated call management entity (ACME). A telephony language was defined and implemented based on the IF-THEN construct to manage the ACME. A graphical user interface was designed to provide a better user interface to the rule-based system. It was found that providing an adequate and conflict free rule set was a complex problem. Difficulties to overcome can be broken down into four parts: timing problems, intra-rule conflicts, inter-rule conflicts and inter-rule set conflicts. Solutions to address the first three problems were proposed. The approaches taken to these problems were to keep the entire ACME system asynchronous, to constrain the rule set by putting constraints in the Phoneditor and to do some post-processing in the ACME to resolve conflicts.

A rule sorting algorithm was developed in the ACME. The sort was performed to order the rules from the most important to the least important. A *tradeoff* was made to give the ACME more *autonomy* to make decisions and act dynamically at the expense of behaving less predictably. To help

alleviate the problem of predictability, a *Show Rules* button was designed in the Phoneditor to display how the ACME is currently interpreting the rule set.

Some proposals were made about how to extend this work and how this work will fit into future telecommunications networks and, in particular, personal communication networks.

Chapter 12

Acknowledgements

I would like to acknowledge the help and support of my supervisor, Chris Schmandt.

I am also indebted to the help of Peter Delaney and his supporting staff in the MIT Telecommunications Department. Their assistance in providing specialized ISDN services and protocol traces has been invaluable throughout whole project.

Ross Snyder, who was a UROP in the Speech Group, and Mick Gardina of Teleos Inc. were a great help when this project began running off an IBM PC. I would like to thank Bill Keats, Debby Hindus and Peter Wong for their help in proof reading this thesis.

Finally, I would like to thank the sponsors of this project - AT&T and Sun Microsystems. In particular, I would like to thanks Ben Stoltz of Sun, who made it possible for us to integrate an ISDN BRI into the Sparestation.

Appendix A

Configuring the Phoneserver

The phoneserver process may execute on two Speech Group machines: **thin-mint** or **shasta**. **thin-mint** is a Sun 386i that runs off of the **chips** file server and **shasta** is a Sparcstation 1 that runs off of the **everest** file server. Using the Yellow Pages services available in SunOS, the names of both these machines are symbolically linked to the word **phoneserver** on their respective file systems. On both machines the source code is in the path `/u/desk/src/isdn/SERVER` while the executable is run from a symbolic link in `/u/desk/bin`. The command line option `-d` will run the phoneserver in debug mode. The `-l` option followed by a DN is used to select the BRI line to be used.

The phoneserver on **thin-mint** is used for production and serves the **pager** and **qdvms** programs described in section 5.3. It has a serial connection to an IBM PC with a B100PC Teleos card that monitors the PDN 34224. The port number used by the phoneserver is resolved in the `/etc/services` file. The token **phoneserver** used by clients to bind to the phoneserver is mapped to port 1400 using Yellow Pages services.

The phoneserver on **shasta** is used for development and runs off of the PDN 88068. The **nhatt** process in `/u/stoltz/PICA` must be running before attempting to run the phoneserver. Again using Yellow Pages services the token **phoneserver_sparc** used by clients to bind to the phoneserver is mapped to port 1500.

When the phoneserver process is started it loads the files `/u/desk/src/isdn/SERVER/dn_to.ca.db.<PDN>` and `/u/desk/calling.name.id` into memory. The first file maps a DN to a CA while the second file maps a DN to a name.

A.1 Enumerating the Event and Command Structures

The event structure is defined in the file /u/desk/src/isdn/event.h. The command structure is defined in the file /u/desk/src/isdn/CLIENT/pserver.h. The structures shown in Figure 5-1 and 5-2 are associated with the define variables shown below.

```
/* for event structure */
#define STATE_LEN 3
#define MAX_CA_LEN 3
#define ISDN_CALL_INFO_LEN 8
#define MAX_DN_LEN 20
#define MAX_NAME_LEN 48
#define LOCAL_DN_LEN 6

/* for command structure */
#define CMD_SIZE 2
#define MAX_CA_LEN 3
#define MAX_SRC_LEN 20
#define MAX_DEST_LEN 20
```

A.2 Call Appearance States

```
#define UNKNOWN_STATE -1
#define WILD_STATE -2
#define NUM_STATES 13
#define REG_idle 0x0001
#define REG_held 0x0002
#define REG_active 0x0004
#define REG_incoming 0x0008
#define REG_dialing 0x0010
#define REG_activated 0x0020
#define REG_deactivated 0x0040
#define REG_pending 0x0080
#define REG_local_hold 0x0100
```



```

#define REG_remote_hold 0x0200
#define REG_confirmed 0x0400
#define REG_ars 0x0800
#define REG_rejected 0x1000
#define REG_outgoing 0x2000

```

A.3 Interest Structure

At startup time, the phoneserver initializes an array of pointers, called `ca_pt`, with each pointer pointing to an empty linked list (shown in Figure A-1) representing the interest of a client. Since the CA field in the Supplementary Voice Service protocol is only two bytes, the maximum array size is 99. The first two fields `entity` and `isdn_channel` are used when communicating with nhatt. The next two fields, `current_state` and `requested_state`, hold the state of the call appearance, which is assumed to be `idle2` at startup time. The field `to_msg_list` is a pointer to a list of all the interest of the client.

```

struct ca_to_linklist {
    int entity;
    int isdn_channel;
    char current_state[STATE_LEN];
    char requested_state[STATE_LEN];
    msgsock_list *to_msg_list;
    struct ca_to_linklist *next;
}

```

Figure A-1: Call Appearance to Interest Structure

Upon receiving a `register_interest` command from a client, the phoneserver converts the DN into a CA and uses the CA as an index to the `ca_pt` array. It then creates interest structures shown in Figure A-2.

Starting with the `msgsock_list` structure, the `msgsock` field contains the socket number, identifying the client, that was used to register an interest. The `gqi` field points to the `interest_list` that actually contains the interest of the client. The `msgsock_list` field is a pointer to link the list since more than one client can register interest in the same CA. Finally, the `interest_list` contains the CA, the number of CAs, the interest and a pointer to link the list. The interest value is a masked value of any one or combination of the call appearance states shown in Figure A.2.

When an event occurs, the CA for that event is obtained and used as an index in `ca_pt`. The state of the event is extracted and compared to the `interest` field in the `interest_list` structure. If

```

struct msgsock_list {
    int msgsock;
    interest_list *gqi;
    struct msgsock_list *next;
}

struct interest_list {
    char get_ca[MAX_CA_LEN];
    int quantity;
    int interest;
    struct interest_list *next;
}

```

Figure A-2: Interest Structures

there is a match, then the entire event structure is sent to the client identified by the *msgsock_list* field in the *msgsock_list* structure.

Appendix B

Configuring the ACME

Before executing ACME, users should set up a configuration file and define an environment variable called `MY_DN` to be the DN they wish ACME to manage. If `MY_DN` is not defined the user may enter it as a command-line option or else he will be asked to enter it by ACME. `MY_DN` is also used as an extension to the file `.acme_rules.<DN>`, shown in section B.1, in the root directory of the user. If this file does not exist, ACME will create one with the key word `begin` at the top but no rules. The configuration file, shown in section B.2, is used when a user wishes to manage more than one DN from a single machine. The file maps a DN into a relative port number used by the ACME to communicate with the Phoneditor. If no configuration file exists, the default service name to bind to is `acme_server0`. There are three ports that are dedicated to support ACME to Phoneditor communications - `acme_server0`, `acme_server1` and `acme_server2`. Using `/etc/services` on the machine `everest`, they are bound to ports 1450, 1451 and 1452 respectively.

ACME can be executed from the `/u/desk/bin` directory. The line level options are as follows. The `-u` option followed by a DN is used to specify the DN of interest. The `-l` option will print on the screen the rule fired by ACME without actually having ACME execute the action. The `-d` option puts ACME into debug mode.

B.1 Sample Rule Set File

File name: `.acme_rules.<DN>`

```
begin
```

```
IF the location of me is my_office THEN transfer the call to where I am  
after 1 seconds
```

IF the calling party is not friends THEN transfer the call to voice mail after 1 seconds

IF the calling party is speech-group THEN transfer the call to 8-8670 after 1 seconds

IF the calling party is anyone and the location of ccwong is Garden THEN transfer the call to voice mail after 10 seconds

IF the calling party is anyone and the time is before 12:00 THEN transfer the call to where I am after 1 seconds

IF the calling party is someone I recently called THEN transfer the call to where I am after 1 seconds

IF the calling party is anyone THEN transfer the call to voice mail after 1 seconds

IF the time is after 18:00 THEN transfer the call to where I am after 0 seconds

IF the location of ccwong is 352 THEN transfer the call to 8-8670 after 1 seconds

IF the status of my phone is active THEN do nothing

B.2 Configuration File

Each line has a DN followed by an ACME port number of 0, 1 or 2.

File name: .acme_config

38026 0

88670 1

B.3 Alias Files

Alias files reside in the root directory of the user. System wide alias files reside in the path /u/desk/data. These files are also used by the *Phoneditor* to include the aliases in the *option* button within the dialog box.

The syntax of the alias files is as follows. Each line must begin with the token *alias*. The alias

name immediately follows the `alias` token and than the alias list. Each member of the list must be delimited by a coma.

```
File name: .acme_rooms
```

```
alias my_office 352
```

```
File name: .acme_group_names
```

```
alias friends Phil, Bill, Marilyn, Carl, Angela, Stephanie
```

```
alias important-person Geek, Marilyn, Phil
```

B.4 Protocol between ACME and the Phoneditor

The Phoneditor can send three commands to ACME: `update`, `sleep` and `wake`. They are defined ASCII strings listed below. ACME will either acknowledge the successful completion of the commands or if the command fails. ACME will send back a negative acknowledgement.

```
/* Phoneditor to ACME messages */
#define UPDATE    "up" /* request ACME to update rule set */
#define WAKE      "wa" /* request ACME to wake up */
#define SLEEP     "sl" /* request ACME to go to sleep */

/* ACME to Phoneditor messages */
#define ACK       "ack" /* acknowledge the update request */
#define ACK_WAKE "acw" /* acknowledge the wake request */
#define ACK_SLEEP "acs" /* acknowledge the sleep request */
```

Appendix C

Lex and Yacc Description of Telephony Language

C.1 Lex File – token.l

```
[0-9:]+ return(TIME);
[0-9-]+ return(DN);
"the person" return(A_PERSON);
"anyone" num_of_anyones++; return(ANYONE);
"and" and_flag = TRUE; return(AND);
"begin" return(BEGIN_TOKEN);
"my calendar" return(CALENDAR);
"pick up the call" return(CALL_PICKUP);
"the calling party is" return(CALLING_PARTY);
"convey the following message" return(CONVEY_MSG);
"drop the call" return(DROP);
"do nothing" return(DO_NOTHING);
"hold the call" return(HOLD);
"if" return(IF);
"IF" return(IF);
"is in" return(IS_IN);
"log the call" return(LOG);
"is logged in" yylval = 1; return(MACHINE_STATUS);
```

```

"is logged out" yylval = 0; return(MACHINE_STATUS);
"I'm away from my office" return(MY_ACTIVITY);
"not" not_flag = TRUE; return(NOT);
"from off campus" return(OFF_CAMPUS);
"from on campus" return(ON_CAMPUS);
"after" op_var = O_AFTER; return(OPERATOR);
"around" op_var = O_AROUND; return(OPERATOR);
"before" op_var = O_BEFORE; return(OPERATOR);
"or" return(OR);
"page me" return(PAGE);
"on hook" return(PHONE_STATUS);
"off hook" return(PHONE_STATUS);
"someone I recently called" return(RECENTLY_CALLED);
"says that" return(SAYS);
"second" return(SECONDS);
"seconds" return(SECONDS);
"the status of" return(STATUS);
"the machine" return(THE_MACHINE);
"my phone is" return(MY_PHONE_IS);
"the time is" return(THE_TIME);
"then" return(THEN);
"\nthen" return(THEN);
"THEM" return(THEN);
"\nTHEM" return(THEN);
"transfer the call to" return(TRANSFER);
"voice mail" return(VMAIL);
"where I am" return(WHERE_I_AM);
"notify me by a pop-up window" return(WINDOW_NOTIFICATION);
\n return ('\n');
quit return 0;
[a-zA-Z-_]+ return(NAME);
[a-z]+ return(MACHINE);
{qstring} return (QSTRING);
. ;

```

C.2 Yacc File – parser.y

```
%token A_PERSON
%token AFTER
%token AND
%token ANYONE
%token BEGIN_TOKEN
%token CALENDAR
%token CALL_PICKUP
%token CALLING_PARTY
%token CONVEY_MSG
%token DN
%token DO_NOTHING
%token DROP
%token HOLD
%token IF
%token IS_IN
%token LOG
%token MACHINE
%token MACHINE_STATUS
%token MESSAGE
%token MY_ACTIVITY
%token NAME
%token NOT
%token OFF_CAMPUS
%token ON_CAMPUS
%token OPERATOR
%token OR
%token PAGE
%token PHONE_STATUS
%token QUOTATION
%token QSTRING
```



```
%token RECENTLY_CALLED
%token SAYS
%token SECONDS
%token STATUS
%token THE_MACHINE
%token MY_PHONE_IS
%token THE_TIME
%token THEN
%token TIME
%token TRANSFER
%token VMAIL
%token WHERE_I_AM
%token WINDOW_NOTIFICATION
```

```
lines: /* empty */
| lines line ;
```

```
line: '\n'
| BEGIN_TOKEN
| IF cond THEN action '\n' ;
```

```
cond:  cond_elem OR cond
      | cond_elem AND cond
| cond_elem NOT cond
| cond_elem ;
```

```
cond_elem: CALLING_PARTY NOT calling_party_id
| CALLING_PARTY calling_party_id
| A_PERSON name IS_IN location
| THE_TIME OPERATOR TIME
| STATUS THE_MACHINE NAME MACHINE_STATUS
| STATUS MY_PHONE_IS PHONE_STATUS
| CALENDAR SAYS MY_ACTIVITY ;
```

calling_party_id: RECENTLY_CALLED

| ANYONE
| DN
| ON_CAMPUS
| OFF_CAMPUS
| NAME ;

action: TRANSFER place delay

| CALL_PICKUP
| DROP
| HOLD
| LOG
| WINDOW_NOTIFICATION
| PAGE
| CONVEY_MSG QSTRING
| DO_NOTHING ;

place: WHERE_I_AM

| DN
| VMAIL
| NAME ;

delay: OPERATOR TIME SECONDS ;

location: TIME

| NAME ;

name: NAME ;

Bibliography

- [1] T Bowen, F Dworack, C Chow, N Griffeth, G Herman, and Y Lin. Feature interaction problem in telecommunications systems. In *Conference on Software Engineering for Telecom Switching Systems*, 1989.
- [2] I Bowles, L Brunet, R Eckert, K Emami, R Kamel, and P Momtaham. Px: Integrating voice communications with desktop computing. *Journal of the American Voice I/O Society - Desktop Audio Issue*, 9:1-19, 1991.
- [3] C Chow, D Braun, and M Adachi. An example in connecting isdn with the intelligent network and the local area network. *Bellcore internal paper*.
- [4] D Cox. Portable digital radio communications - an approach to tetherless access. *IEEE Communications Magazine*, pages 30-40, 1989.
- [5] D Cox. Personal communications - a viewpoint. *IEEE Communications Magazine*, pages 8-20, 92, 1990.
- [6] J Gilmour and R Gove. Intelligent network/2: - the architecture - the technical challenges - the opportunities. *IEEE Communications Magazine*, pages 8-11, 1988.
- [7] R Hass and R Humes. Intelligent network/2: A network architecture concept for the 1990s. In *ISS '87. A12.1*, 1987.
- [8] G Herman, M Ordun, and C Riley. Between laboratory and field trial: Experience with a communications services testbed. In *Proceedings of the Human Factors Society - 30th Annual Meeting*, pages 804-808, 1986.
- [9] G Herman, M Ordun, C Riley, and L Woodbury. The modular integrated communications environment (mice): A system for prototyping and evaluating communications services. In *Proceedings of International Switching Symposium '87*, pages 442-447, Phoenix, Arizona, 1987.

- [10] G Herman and C Riley. Services for the next generation network: Experience with a network service testbed. In *Proceedings of the First European Conference on Information Technology for Organisational Systems - EURINFO '88*, pages 1167-1172, 1988.
- [11] Forward is an internal Speech Group phoneserver application written by Chris Schmandt.
- [12] The Socket Manager is an internal Speech Group tool written by Barry Arons.
- [13] QDVM is an internal Speech Group voice mail system written by Barry Arons, Sanjay Manandhar, Lisa Stifelman, and Chi Wong.
- [14] R Kamel, K Emami, and R Eckert. Px: Supporting voice in workstations. *IEEE Computer*, 23:73-80, 1990.
- [15] S Manandhar. Activity server: A model for everyday office activities. Master's thesis, Massachusetts Institute of Technology, June 1991.
- [16] S Manandhar. Activity server: You can run but you can't hide. In *to appear in Usenix Summer 1991 Technical Conference*, June 1991.
- [17] N Matsuo, K Shimohara, H Matsui, and Y Tokunaga. Personal telephone services using ic-cards. *IEEE Communications Magazine*, pages 41-48, 1989.
- [18] C Riley. Experiences with an integrated voice and text message service. In *Proceedings of the Human Factors Society - 31st Annual Meeting*, 1987.
- [19] R Root and C Koster. Experimental evaluation of a mnemonic syntax for controlling advanced telecommunications services. In *Proceedings of the Human Factors Society - 30th Annual Meeting*, pages 809-813, 1986.
- [20] R Root and C Koster. Experimental evaluation of a mnemonic command syntax for controlling advanced telecommunications services. In *Proceedings of the Human Factors Society - 31st Annual Meeting*, 1987.
- [21] C Schmandt and B Arons. Desktop audio. *Unix Review*, October 1989.
- [22] C Schmandt and S Casner. Phonetool: Integrating telephones and workstations. In *Proceedings, GLOBECOM '89*. IEEE Communications Society, November 1989.
- [23] W Stallings. *ISDN : an Introduction*. Macmillan, New York, 1989.
- [24] R Steele. Deploying personal communications networks. *IEEE Communications Magazine*, pages 12-15, 1990.
- [25] Teleos Communications, Inc., Eatontown, NJ. *ASK100 Access Systems Kit User Manual*, 1989.

[26] S Tufty. Watcher. *MIT Bachelor's Thesis*, 1990.

[27] P Zellweger, B Douglas, and D Swinehart. An experimental environment for voice system development. *IEEE Office Knowledge Engineering Newsletter*, 1987.