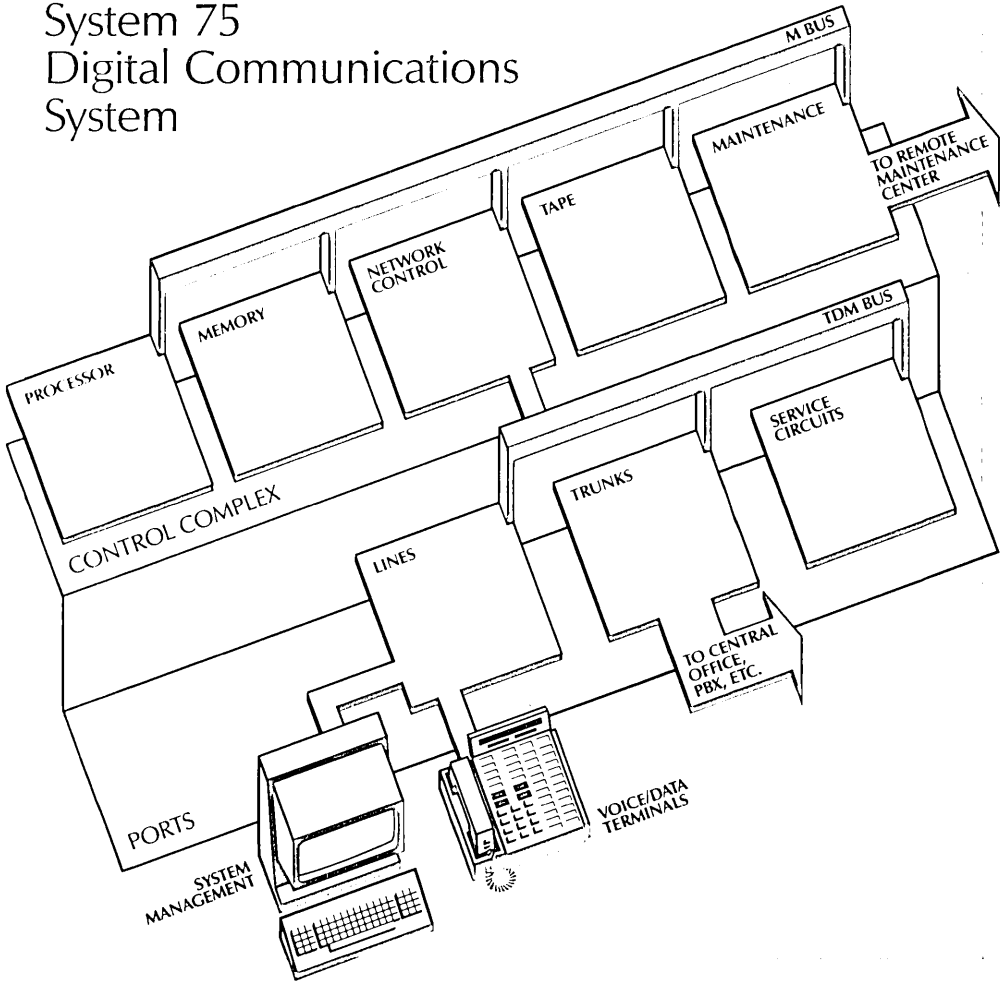
January 1985 Vol. 64 No. 1 Part 2

AT&T TECHNICAL JOURNAL

A JOURNAL OF THE AT&T COMPANIES

System 75 Digital Communications System



AT&T TECHNICAL JOURNAL (ISSN 8756-2324) is published ten times each year by AT&T, 550 Madison Avenue, New York, NY 10022; C. L. Brown, Chairman of the Board; T. O. Davis, Secretary. The Computing Science and Systems section and the special issues are included as they become available. Subscriptions: United States—1 year $35; foreign—1 year $45.

Back issues of the special, single-subject supplements may be obtained by writing to the AT&T Customer Information Center, P.O. Box 19901, Indianapolis, Indiana 46219, or by calling (800) 432-6600. Back issues of the general, multisubject issues may be obtained from University Microfilms, 300 N. Zeeb Road, Ann Arbor, Michigan 48106.

Payment for foreign subscriptions or single copies must be made in United States funds, or by check drawn on a United States bank and made payable to the AT&T Technical Journal and sent to AT&T Bell Laboratories, Circulation Dept., Room 1E335, 101 J. F. Kennedy Pky, Short Hills, NJ 07078.

Printed in U.S.A. Second-class postage paid at Short Hills, NJ 07078 and additional mailing offices. Postmaster: Send address changes to the AT&T Technical Journal, Room 1E335, 101 J. F. Kennedy Pky, Short Hills, NJ 07078.

AT&T
TECHNICAL
JOURNAL

VOL. 64 JANUARY 1985 NO. 1, PART 2

SYSTEM 75 DIGITAL COMMUNICATIONS SYSTEM

C. D. Weiss, Guest Editor

Introduction and Overview 145
A. Feiner, E. J. Rodriguez, and C. D. Weiss

Communications and Control Architecture 153
L. A. Baxter, P. R. Berkowitz, C. A. Buzzard, J. J. Horenkamp, and F. E. Wyatt

Physical Architecture and Design 175
A. S. Loverde, H. D. Frisch, C. R. Lindemulder, and D. Baker

Switch Services Software 197
W. Densmore, R. J. Jakubek, M. J. Miracle, and J. H. Sun

System Management 213
H. K. Woodland, G. A. Reisner, and A. S. Melamed

Maintenance Architecture 229
K. S. Lu, J. D. Price, and T. L. Smith

The Oryx/Pecos Operating System 251
G. R. Sager, J. A. Melber, and K. T. Fong

Project Development Environment 269
T. S. Kennedy, D. A. Pezzutti, and T. L. Wang

Software Development Tools 287
T. J. Pedersen, J. E. Ritacco, and J. A. Santillo

GAMUT: A Message Utility System for Automatic Testing 305
C. J. Lake, J. J. Shanley, and S. M. Silverstein

Introduction Activities and Results 321
M. A. McFarland and J. A. Miller

ACRONYMS AND ABBREVIATIONS 333

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

Introduction and Overview

By A. FEINER, E. J. RODRIGUEZ, and C. D. WEISS*

(Manuscript received July 11, 1984)

In 1980, a group of system designers at AT&T Information Systems Laboratories (then a part of AT&T Bell Laboratories) was asked to produce a new communications system for the intermediate-size business office (40 to 400 users), to complement a larger system already under development—System 85. The new system was named the System 75 office communication system. Its purpose was to meet the competitive challenge for a high-function digital communication system whose integrated technology could address the evolving needs in office communications and automation. The proposal developed by our designers was based on an all-new hardware and software architecture.

In this intermediate-size range, the existing Feature Package 15 of the *Dimension*® PBX already had provided a challenging standard. Its more than 150 PBX features would have to be included in any new AT&T offering. Like the larger System 85, it would also provide integrated data switching capabilities, including 64-kb/s transparent switching; and simultaneous voice/data transmission using the Digital Communications Protocol (DCP)† that supports two 64-kb/s voice

* All authors are members of AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc.

† Acronyms and abbreviations used in the text are defined at the back of the *Journal.*

and data channels and one 8-kb/s signaling channel at a single interface. With a significant fraction of customers requiring efficient multilocation service, it was recognized that the System 75 switch should also support multilocation networking services available in the current-generation *Dimension* systems and in System 85. These include Distributed Communication System (DCS) operation—permitting significant feature transparency between locations, a DS1 Interface to T1 facilities, Electronic Tandem Network support, and Centralized Attendant Service (CAS). AT&T Information Systems Architecture provided standards with regard to customer-system interactions, terminals, adjuncts and interfaces.

In mid-1983, an internal prototype with partial feature content underwent trials at AT&T Information Systems Laboratories. The first commercial customer received service early in 1984 and the product was publicly announced on April 26, 1984. The system's initial features are listed in the Appendix A to this introduction.

Customer systems have come of age insofar as complexity and sophistication are concerned. Based on the technologies of microprocessors, modern software engineering, and VLSI, products such as System 85 and System 75 have a range and extent similar to the much larger switching systems designed for central office use. Since we felt that there was much of interest to be reported on their design, development, and project methodologies, early in 1984 we decided to produce this special issue of the *AT&T Technical Journal* on System 75. This collection of papers was therefore assembled to serve as an example of modern design in customer communication systems.

There are four major groupings of papers in this issue. The first two papers deal with the switch and control architecture—realized in hardware and firmware—and the physical architecture of the overall product. The next group of four papers describes those functional components realized in software: switch services (a generalization of what is traditionally known as "call processing"); system management, including database aspects and user interface; maintenance for all hardware and software elements; and the real-time operating system. The third group of three papers treats project methodology and software tools upon which the methodology and firmware/software development rests. The first of these papers provides a project management overview and explains how designs from each development community are coordinated and integrated. The tools paper focuses on the software development methodology and tools environment. The test tool is described in a separate paper. Testing was a critical aspect of system development requiring a unique computer-based capability. The final topic deals with bringing the product to the people it serves—customers, and sales and service personnel—those who ultimately judge

the acceptability of the product. The paper describes how the product is introduced to these users, how their reaction is measured, and how corrective measures are taken where necessary.

An important theme throughout these papers is the degree of overall design and development unity and coordination that was required and achieved. Tools, for example, were specified jointly by tools builders and end-user developers. Although an early version of the operating system had been built to support some exploratory call processing development prior to the start of this project, its evolution and optimizations reflected the needs discovered by those developing services for System 75. All maintenance design required significant support in hardware and device design, firmware (on port boards and in the common control) and, of course, in the software. Specification, design, integration, and testing spanned all project areas. System test, performed by a separate test group, depended on an intimate knowledge of software and software-firmware interfaces, and on quick turnaround from developers for fixes to permit testing to progress beyond the curent trouble area. These are but a few examples of intra-project development coordination. The last paper of this issue illustrates the same broad coordination tasks involving all of AT&T Information Systems and its customers.

ACKNOWLEDGMENTS

Only a few of the many people contributing to the design and development of System 75 are authors in this issue. To the other creative and dedicated individuals who had a part in its successful development, we are most thankful and appreciative. Although this group of papers stresses the development aspects of System 75, centered in one organization, the project received outstanding support and product contributions from other AT&T Information Systems and AT&T Bell Laboratories organizations. Specific individuals and the areas in which they led are: D. A. Keller and R. S. Breen (Systems Engineering); D. B. James (architectural consultant); R. S. Berryman (selected software tools); and C. O. Riddleberger (AT&T Bell Laboratories Power Systems).

APPENDIX A

System 75 Features

Standard System Features

Advanced private line termination
Automatic route selection
Code calling access
Direct department calling
Direct inward dialing
Restrictions
  Code restriction
  Inward restriction
  Manual originating line
  Manual terminating line

Direct outward dialing
Emergency transfer
Flexible numbering of stations
Foreign exchange central office access
Intercept treatment
Intercept with lockout
Listed directory number service
Loudspeaker paging
Modem pool
Multiple call appearances of extensions
Multiple listed directory numbers
Music-on-hold access
Off-premises stations
Outgoing facility management
Outgoing trunk queueing
Personal central office line
Primary extension
Recorded telephone dictation access
Remote access
  Miscellaneous trunk restriction
  Origination restriction
  Termination restriction
  Toll restriction
Serial calls
Simultaneous voice/data communication
Station busy indication
System management
  Administration
  Maintenance
  System parameters
  Traffic measurements
Tandem tie trunk switching
Terminal dialing
Through dialing
Tie trunk access
Touch-Tone calling
Touch-Tone sending
Touch-Tone to dial pulse conversion
Uniform call distribution
WATS access

Standard Terminal Features

Abbreviated dialing
Personal, group and system lists
Automatic callback
Bridged call
Call answer from any station
Call coverage
  Caller response interval
  Consult/return
  Coverage information display
  Temporary bridged appearance
Call forwarding—follow me
Call park
Call pickup
Call status indication
Call waiting
Called party identification (with display)
Calling party identification (with display)
Common audible alerting
Conference
Conference/transfer
Data handlers from voice terminal
Dial access to attendant
Dialed number display (with display)
Distinctive alerting
Elapsed time display (with display)
Exclusion
Hold
Hot-line service
Hunting
Intercom
Integrated directory service
I-Use indication
Leave word calling
  Message retrieval (with display)
Manual signaling
Preference
Preselection
Priority calling
Recall signaling
Repertory dialing
Station-to-station calling
Station-to-station-only calling
Time of day and date display
Transfer

Standard Digital Modem Features

EIA RS-232C interface
Speeds up to 19,200 b/s
Synchronous or asynchronous operation (300 or 1200 b/s) for pooled modem operations
Full or half duplex operation
Keyboard dialing (ASCII)
Automatic answer
Automatic speed and mode detection
Automatic and manual self-test
Odd, even or no parity
Other data set like options (e.g., loss of carrier disconnect)

Standard Attendant Specific Features

Alternate console position(s)—up to 6
Attendant call waiting
Attendant direct extension selection with busy lamp field
Attendant display
Central attendant service
Direct trunk group selection
Night service
Release loop operation
Splitting—one way automatic

Class-of-service display
Incoming call identification display
Trunk identification display
Attendant lockout
Attendant transfer—all calls
Splitting—auto-manual
Straightforward outward completion
Trunk group busy/warning indicators to attendant
Trunk-to-trunk connections
Two-party hold on console

Message Center Agent Features

Display shows:
Called person's name and telephone number
Reason for call coverage (busy, no answer, send all calls, go to coverage)
Messages for callers from called person (such as status information)
Calling person's name and telephone number (if internal call)
Can record messages for intended called person or multiple people (causes their automatic message waiting lamp to light) as requested by caller
Has access to directory service to provide caller with additional information (like room number or supervision)

Peripheral Equipment

Terminals
Single line voice terminals
2500
7101 A with two fixed feature buttons
7103A with four fixed feature buttons and 10 programmable feature buttons
Multiappearance voice terminals
7300 Series
7303S with six fixed feature buttons and ten buttons each programmable to either activate features or as call appearances
7305S Same as 7303S plus 24 programmable feature buttons
7400 Series
These terminals provide simultaneous voice and data transmission. Addition of a Digital Telephone Data Module to the 7403D and 7405D models provides an RS-232C interface, allowing the connection of data equipment (data terminals, etc.) to the voice terminal.
7403D
7405D (also supports an optional 40-character numeric display and call coverage module)
Video Terminals
Data Terminal
513 Business Communications Terminal (data only)
Voice/Data Terminals
515 Business Communications Terminals Integrated with digital telephone

Data modules
These modules allow data equipment, such as terminals and computers, to be connected to System 75.
Digital telephone Data Module
Provides an RS-232C interface for data equipment when used in conjunction with a digital terminal (see above)
Processor Data Module
Provides an RS-232C interface to a host computer or standalone terminal
Trunk data module
Provides an RS-232C interface to a private data line or *Digital Data System*® data service unit to a remote computer or terminal

Attendant console
The attendant console includes a 40-character alphanumeric display, command keys, feature status indicators, alarm indicator, and a direct extension selector with busy indicators. The console plugs into any standard telephone wall jack.

Applications processor equipment
500 Business Communications Terminal
A data terminal, with a video display and keyboard connected to an Applications Processor via hard-wired 56-kilobit/second links

Printers
A family of printers is offered to work with applications processing services. The printers have varying speeds, print quality, and cost.
443: Low-speed, draft-quality matrix printer
445: Medium-speed, draft-quality line printer
460: Medium-speed, draft-quality matrix printer
450: Low-speed, reproduction-quality printer
Station Message Detail Reporting (SMDR)
Processing Options
COMMSTOR II & *Teleseer*® series
Stores details of all calls made and does SMDR processing using tariff tables
Local Storage Unit (LSU)
Stores details of all calls made for MDR processing
Applications Processor
The Call Detail Recording and Reporting feature provides SMDR processing on the associated System.
Printers
Local associated printer may be used to print formatted data

System Access Terminal
513 Business Communications Terminal
Optional 470 printer

APPENDIX B

System 75 Specifications

Switch Cabinet
70"h × 32"w × 24"d (large)
42½"h × 32"w × 24"d (small)

System Limits: (First Release)

Time Slots	512
Circuit Switch	64 Kb/s
Calling Rate	1800/hr
Traffic Limit	7200 CCS
Stations	400
Data Modules	200
Trunks	200
Trunk Groups	50
Pooled Modems	32
Attendant Consoles	7

Cabling Limits:

Analog	6000 ft.
Hybrid, MET	1000 ft.
Digital, Data Modules, 515 BCT	3400 ft.
513 BCT	5000 ft.

Thermal Output
Maximum—1250 watts (4250 BTU's/hr)
Typical Average—875 watts (3000 BTU's/hr)
[Cabinet is equipped with forced air cooling]

Power Requirements: 115V 60 Hz 50A
Dedicated unswitched outlet located within 10 feet. Approved grounding essential.

Environment:
Temperature—40°–110°F
Relative Humidity—10–95% up to 78°, decreasing to 35% at 110°
Well-ventilated area free of corrosive gasses and excessive dust or dirt.

AUTHORS

Alec Feiner is the Executive Director at AT&T Information Systems responsible for the development of office automation products. He started his engineering career with Bell Laboratories in 1953 and was a member of the team that developed the first stored program controlled electronic switching systems. Since 1969 he has been involved in customer premises telephony contributing toward the creation of numerous systems, among them, *Dimension*® PBX, *Horizon*® communication system, and System 75. A graduate of Columbia University, Feiner holds 40 patents and is the author of numerous articles on switching. He was awarded the Bell Laboratories Fellow Award in 1982 and in 1983 was elected to the National Academy of Engineering.

Ernesto J. Rodriguez, B.S. (Mathematics), 1967, Michigan Technological University; M.S.O.R., 1970, New York University; Bell Laboratories, 1967–1983; AT&T Information Systems Laboratories, 1983—. At Bell Laboratories, Mr. Rodriguez worked on computer access for *Picturephone*® service and was involved in developing software planning aids for the Digital Data System. He was also responsible for the software design and development of the Transaction Network and the Bell Packet Switching Network. Presently, Mr. Rodriguez is a Director at AT&T Information Systems Laboratories with responsibility for System 75 development.

C. Dennis Weiss, B.S. (Electrical Engineering), 1961, Stanford University; M.S. (Electrical Engineering), 1962, Columbia University; Ph.D. (Electrical Engineering), 1966, Columbia University; Bell Laboratories, 1972–1982; AT&T Information Systems Laboratory, 1983—. Mr. Weiss taught at Johns Hopkins University from 1966 to 1972. At AT&T, he has been involved with exploratory and development work on business communications systems. Currently, he is Director of the Integrated Systems Laboratory, responsible for System 75 development.

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

Communications and Control Architecture

By L. A. BAXTER,* P. R. BERKOWITZ,* C. A. BUZZARD,†
J. J. HORENKAMP,* and F. E. WYATT*

(Manuscript received July 11, 1984)

The System 75 office communication system uses a unique communications and control architecture that provides great flexibility and a minimum of overhead for small configurations while growing smoothly to larger line sizes. A distributed communication network provides 64 kb/s connectivity for both voice and data. It consists of a pair of time division multiplexed buses and intelligent port circuits. Flexible conferencing and gain adjustment are supported as an integral part of the network. The control complex supports an operating-system-based software structure.

I. INTRODUCTION

The System 75 office communications system hardware architecture consists of a control complex and a communications network, which are connected by a pair of Time Division Multiplexed (TDM)‡ buses, as shown in Fig. 1. Part of the bandwidth of the TDM buses is used as a control channel between the control complex and the intelligent port circuits in the communications network.

* AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc. †AT&T Information Systems Laboratories; present affiliation, Bell Communications Research, Inc.

‡Acronyms and abbreviations used in the text are defined at the back of the *Journal.*

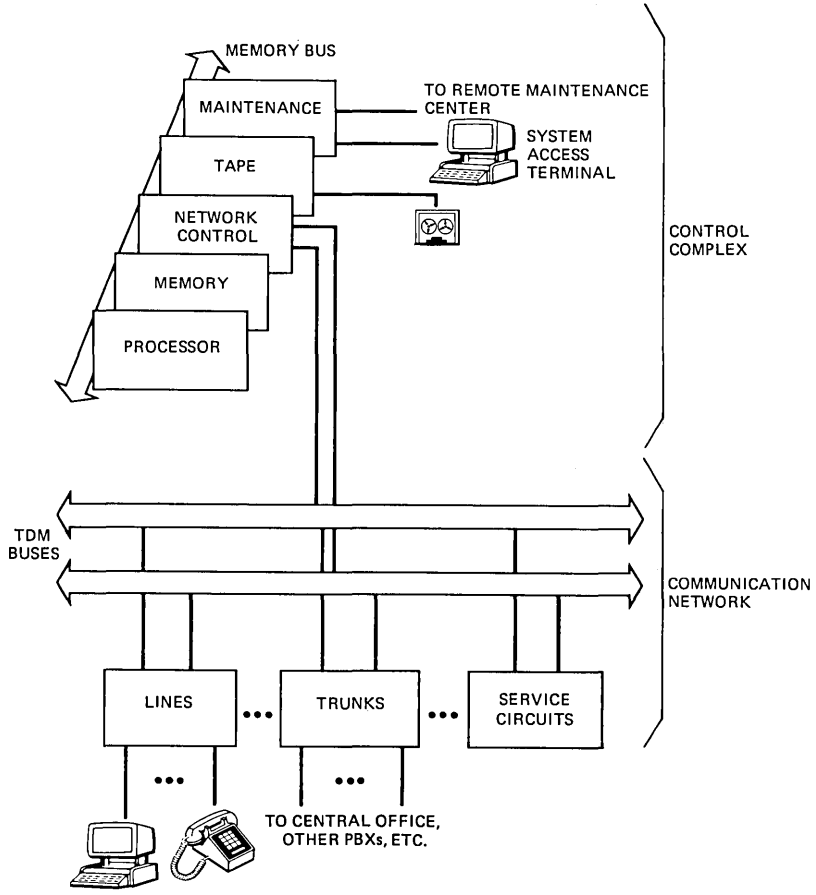


Fig. 1—System 75 communication and control architecture.

One of the main architectural features of System 75 is the distributed switching network. This was chosen to allow as much complexity as possible to be transferred to the port boards, thereby reducing the amount of common circuitry required and minimizing the getting-started cost (often referred to as the intercept cost). At the same time, aggressive use of VLSI devices in the port circuits allows the per-port cost (slope) to remain low. This combination of low intercept cost and moderate slope allows the System 75 network to remain cost-effective over a wide range of line sizes.

Another key architectural feature is the universal slot concept. All circuit pack slots in System 75 port carriers are identical (with the exception of a unique address for each slot). Every port slot has the same interfaces to the TDM buses, I/O access to outside devices, and

power supplies. This freedom to plug any port circuit into any slot allows customers to flexibly configure their system, so that it is optimized to their particular needs.

This highly distributed and modular architecture has allowed System 75 to meet the following design objectives:

1. Provide a digital network which efficiently supports both voice and data communication.
2. Serve customers up to four hundred lines with a single-cabinet hardware configuration.
3. Provide support for an operating system-based software structure.
4. Maintain high-reliability operation with complete self-diagnosis and alarming.
5. Utilize a flexible architecture so that future needs may be easily accommodated, and the system may be gracefully upgraded as technology advances.
6. Provide the above functions at a competitive price.

The following sections describe the architecture in more detail.

II. COMMUNICATION NETWORK

2.1 TDM buses

System 75 has two parallel TDM buses, each of which is 8 bits wide and runs at 2.048 MHz. Functionally, the dual-bus structure is equivalent to a single 512-time-slot bus. Separating the bandwidth into two physically distinct buses has two advantages. First, it lowers the speed of each bus, which eases the timing requirements on VLSI interface devices. Second, the redundancy provided by two buses improves system reliability. If one bus fails, the architecture permits continued operation at reduced capacity on the other bus.

The buses are implemented as printed paths on the backplane. The geometry of these printed paths has been carefully designed to maintain the proper characteristic impedance. Several carriers may be daisy-chained together within a cabinet. Each bus path has a resistive termination at each end.

A novel current-source bus transceiver was designed for this application. Up to 100 port boards may be plugged into the bus in a simple party-line fashion. These transceivers have been specifically designed with low signal levels and controlled rise times to minimize radiated noise. They are designed to allow nondisruptive insertion and removal of boards with the system power on and to isolate port boards from the bus during failure conditions.

Voice signals on the buses are encoded in μ-255 PCM (Pulse Code Modulation) format[1] for domestic systems, while A-law PCM could be provided for international applications. Data signals utilize the Digital

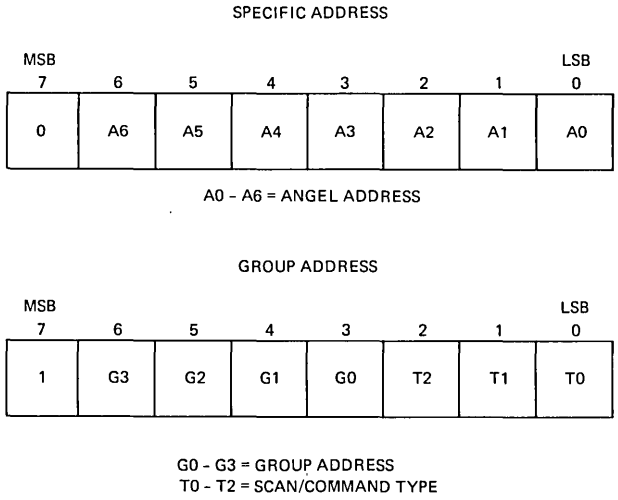


Fig. 2—Addressing modes of TDM bus control channel.

Communications Protocol (DCP).[2] Several time slots are reserved for tone distribution and for the control channel.

2.2 Control channel

The first five time slots on each bus are reserved for a control channel between the control complex and the port circuits. In essence, the control channel is the backbone for a network of microprocessors. It provides a communication path between the control complex and the microprocessor on each port circuit pack (commonly referred to as the angel). On each port board a custom VLSI device known as the SAKI (Sanity and Control Interface) provides address recognition, buffering, and synchronization between the angel and the five control time slots. The control channel is active on only one bus at a time. It can be moved to the other bus in the event of a bus failure.

The control channel operates strictly in a polled mode, with the network control (often referred to as the archangel) as master, and the angels as slaves. The first time slot of each frame (TS0) carries the control address, while the following four time slots (TS1 to TS4) contain control data. The archangel grants bus usage to a particular angel or group of angels by transmitting a specific address in TS0. The direction of transmission during the control data time slots (TS1 to TS4) is dependent on the message type.

Each port slot in System 75 contains seven address pins that are hard-wired to define a unique address. During initialization, the angel reads in this address and writes it to the address-detection portion of the SAKI. This seven-bit address fixes a limit of 127 angels (plus one

null address) in the archangel's address space. This restriction is well above the physical limitations on the number of port circuit packs in a single cabinet.

There are two modes in which the archangel can address an angel, as shown in Fig. 2. To differentiate between these two modes, the SAKI inspects the Most Significant Bit (MSB) of the control address which appears in TS0. If the MSB = 0, then the archangel is addressing the specific angel whose address is given in the remaining seven bits. If the MSB = 1, then the archangel is addressing the group of eight angels whose address matches bits 3 through 6 of TS0. In this case, the three Least Significant Bits (LSB) indicate the type of scan or command.

2.2.1 Group addressing

The group address mode is used in two ways: to collect status information (called short-scanning) and to send certain commands to a group of eight angels simultaneously (group commands). In a short scan, each angel in the addressed group responds with a single bit of information in TS2. The bit-position assignments on the TDM bus are determined by a binary decoding of the lower three bits of the angel address. (In other words, the angel whose lower three address bits are 000 responds on TDM bus bit 0, etc.) Using short scans, the archangel can gather status information from a full complement of angels an order of magnitude faster than if each angel were polled directly. Thus, short scanning reduces the latency period before a stimulus is reported to the control complex.

The archangel obtains two types of status information via short scans. Activity scans determine which angels have messages waiting for uplink transmission. Those angels will be individually polled to collect the messages. Sanity scans collect state-of-health information from the angels.

The sanity control circuitry in the SAKI gives the control complex the ability to identify and isolate insane angels quickly. When the archangel sends a sanity scan to a group of eight angels, each SAKI in the group checks the sanity of its angel by verifying that its angel has updated a special sanity bit latch in the SAKI since the previous sanity scan. If its angel has cleared the sanity bit, the SAKI notifies the control complex of its angel's health by driving its bit low during TS2. If the angel has not cleared the sanity bit, indicating angel insanity, the SAKI sends no response in TS2, notifying the Switch Processing Element (SPE) of the angel's insanity. Simultaneously, the SAKI resets its angel, holding it idle, and forces the bus transceivers into a receive-only mode, preventing the port board from errantly transmitting onto the TDM bus. The SAKI waits for the restart

instruction from the control complex before allowing the angel to begin running again.

Unlike traditional watchdog circuits where each processor must update a local timing circuit periodically to indicate sanity, System 75's sanity control gives the control complex total control over the sanity scanning rate, the number of times an insane angel is restarted, and the ability to shut down an individual angel at will.

Each SAKI also protects the control channel by monitoring transmission onto the TDM bus during the control time slots. When it detects transmission during a control time slot by anyone other than itself, it disables its angel and Network Processing Elements (NPEs) and waits for the control complex to send the restart command.

2.2.2 Specific addressing

When the control complex wants to send a message to a specific angel, or retrieve a message from an angel that gave a positive response to an activity scan, the archangel must use the individual addressing mode. In this mode, a message may span a number of frames.

Messages sent across the control channel use a well-defined format known as the Control Channel Message Set (CCMS). The CCMS provides a combination of stimulus and functional messages that are common across all types of ports. Downlink (network control to port circuit) messages allow the control complex to control the ringer and LEDs on stations; seize, release, and outpulse on trunks; set up and tear down network connections; execute various maintenance tests, etc. Uplink (port circuit to network control) messages allow the port to report state changes, such as switchhook and button pushes on stations and seizure of incoming trunks. Control channel messages are protected by a checksum, and are retransmitted in the event of an error.

2.3 Network processing element

2.3.1 Switching functions

In conventional digital switches, each port is permanently assigned a time slot on which to talk and another on which to listen. A centralized mechanism called a Time Slot Interchanger (TSI) is used to enable, reorder, and transfer time slots from talking to listening ports. Conventional TSIs require additional centralized equipment to perform gain adjustment or form conferences, features that require arithmetic processing of voice samples. However, intelligent TSIs have been designed to perform these operations.[3]

Whether intelligent or not, a centralized TSI that is sized to accommodate the full capacity of the communications system represents a cost burden on small-size customers who pay for more capacity than

they need. The alternative of providing a family of TSIs optimized for several sizes entails extra development effort and complicates growth in the field.

The communications network in System 75 solves these problems. Each port board carries with it a modular piece of the network in the form of a VLSI chip, the NPE.[4] Each NPE serves four ports and is resident on each of the port boards. Two are used on each of the eight-port boards with the exception of the digital line, which uses four NPEs to switch the two information channels of each of its eight ports. The distributed network architecture and absence of a centralized TSI allows customers to buy just the right amount of network for their needs and permits smooth growth as needs expand, while the VLSI technology provides a low per-port cost even though each port has its own dedicated TSI and voice processing logic.

The NPE provides the functions of time-slot assignment for listening and talking, gain adjustment, and eight-party conferencing.[5] (System 75 actually features a six-party conference limit with the remaining two conferencing slots reserved for tones.) The NPE contains over 18,000 transistors, with half of them making up a novel memory network for control and processing functions. As an indication of its complexity, a Transistor-Transistor Logic (TTL) breadboard of this device required six 8- by 13-inch circuit boards.

2.3.2 NPE operation

Figure 3 is a diagram of the operation of one of the NPE's four channels. A network of memory arrays, the associative conference buffer,[6] is used both as a control store, written and read by the angel, and as a buffer for PCM samples from the time division bus. Memory locations are loaded by the angel with time-slot numbers for specifying

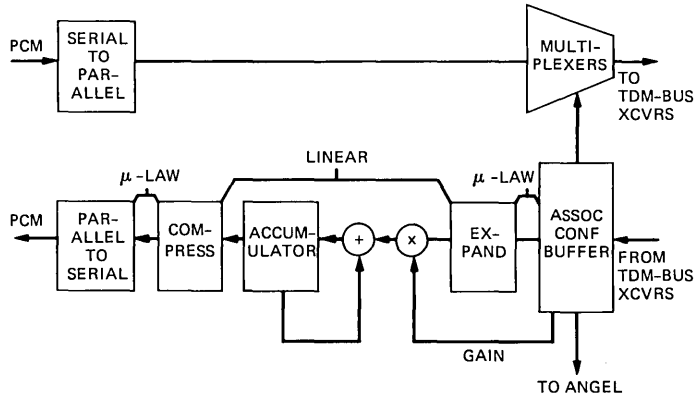


Fig. 3—Architecture of one of the network processing element's four channels.

a talking slot and up to seven listening time slots. Companion memory locations are loaded with a gain or loss value to be applied to samples from listening time slots received from the bus. A talk enable/disable bit can also be stored for the talking time slot. The locations holding the time slot number also act as a content addressable memory by comparing their content against a time-slot counter and individually controlling a sample transfer on the specified time slot. The sample transfers consist of placing a talking sample from a station onto the bus and storing up to eight listening samples from the bus in a sample buffer memory array. The sample buffer holds the samples until they are accessed with their respective gain values to form a conference sum.

An active port is usually allowed to talk on one time slot and listen to from one to seven others. An idle port uses no time slots. When multiple listen time slots are selected, their samples are converted to linear PCM, multiplied by the stored gain values, summed together in an accumulator, and then restored to μ-law for delivery to the station. Of course, data samples are passed through the NPE without any of the conference processing which would corrupt the data.

A simple two-party connection occupies two time slots: each port talks on one and listens to the other. An N-party conference uses N time slots. Tones may be broadcast on a single time slot and received by an unlimited number of ports. In System 75 the seven single-frequency components of the Dual-Tone Multifrequency (DTMF) signaling tones are broadcast continuously. Each port requiring access to a DTMF tone for dialing out forms a brief conference between the two appropriate single frequencies without interfering with other ports similarly doing so.

2.3.3 Conferencing algorithms

The forming of a gain-adjusted conference sum can be thought of as a sequence of arithmetic operations. (In the following discussion, conversions between linear and μ-law PCM are neglected, since they are required in all cases.) Each recipient of the conference sum must hear the composite of the other conferees' samples minus his or her own. The sample received by the kth member of an N-party conference is:

$$R_k = \sum_{i=1}^{N} g_{ik}T_i - g_{kk}T_k,$$

where

R_k = Receive sample for port k
T_k = Transmit sample for port k
g_{ik} = Gain coefficient from port i to port k

T_i = Transmit sample for port i

g_{kk} = Gain coefficient from port k to itself.

There are conceptually two algorithms for generating a conference sum. One method is to hold the g_{ik} constant for all k. Then:

$$R_k = \sum_{i=1}^{N} g_i T_i - g_k T_k,$$

where g_i = Transmit gain for ith port. First a conference sum of all gain-scaled transmit samples is formed, a task requiring N multiply and accumulate operations. Then the receive samples are generated by subtracting out the receiving port's scaled transmit sample, an additional N operations, for a total of $2N$ operations per conference.

The chief advantage of the "$2N$" algorithm is its efficient execution, an important property in a centralized, intelligent TSI where processing throughput may be a constraint. A disadvantage is the inflexibility it imposes on conference call transmission gains, since all conferees must listen to a given port with the same gain. Often this results in loss in excess of that required for stability or optimum intelligibility.

The second method allows individually chosen interport gains between all conferees but requires considerably more processing effort for large conferences. It builds each receive sample R_k separately:

$$R_k = \sum_{\substack{i=1 \\ i \neq k}}^{N} g_{ik} T_i.$$

Samples from the other N-1 conferees are individually multiplied by the appropriate transmission gain coefficient and added to the partial sum as it is built up. This requires N-1 multiply operations. Since the sum must be formed separately for each receiving port (a total of N times), $N \times (N - 1) = N^2 - N$ operations are needed to form the conference. The advantage of this "N-squared" algorithm is the freedom it allows in choosing transmission gain coefficients. Each party can listen to the other conferees with an individually tailored gain.

System 75 achieves an N-squared conference algorithm since each of the N NPEs in a conference perform the required N-1 operations with individually chosen gain coefficients. The necessary processing throughput is obtained as a natural benefit of the parallelism inherent in the NPE-based network.

The importance of N-squared conferencing is illustrated by a three-way call involving two telephone line ports and a central office trunk port. PBX line ports optimally require about 6 dB of loss between them to simulate losses normally encountered in the loop plant, while trunk connections should have 0-dB transmission loss between them,

as shown in Fig. 4. This combination of loss relationships can only be achieved with an N-squared algorithm since a $2N$ method would subject the trunk to the same 6-dB loss that is applied between lines. The NPE implementation of this three-party conference is illustrated in Fig. 5.

2.4 Intelligent port circuits

Figure 6 is a block diagram of a generic System 75 port circuit. The

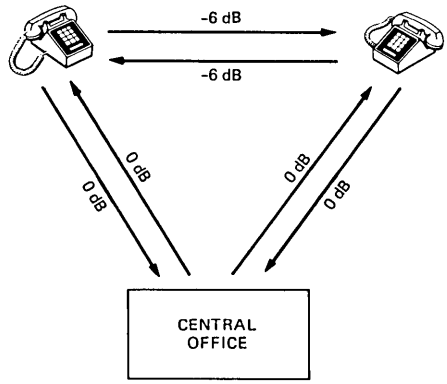


Fig. 4—Example of System 75 gain plan for three-party conference.

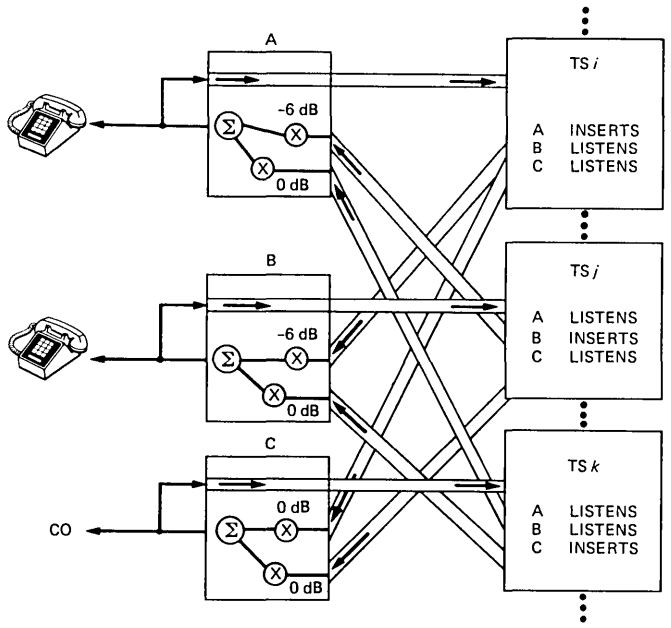


Fig. 5—System 75 implementation of three-party conference.

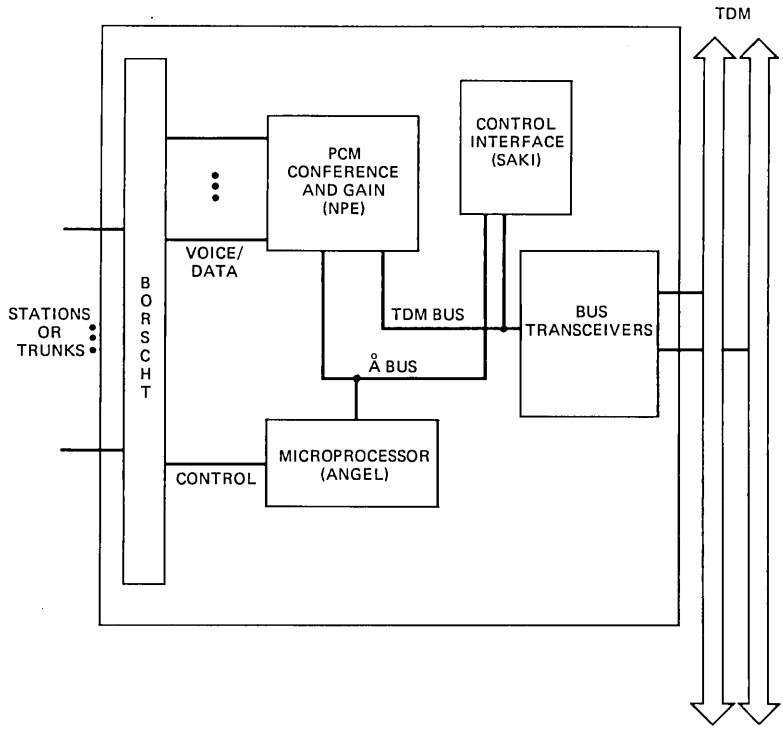


Fig. 6—Architecture of a System 75 generic port board.

circuit pack interfaces to the TDM buses via the custom bus transceivers. Time-slot information, which may be either PCM voice samples or data, is handled by the NPEs (Section 2.3). The interface to the control channel is handled by another VLSI device, the SAKI (Section 2.2).

The BORSCHT circuitry contains whatever is necessary to interface to a particular type of line or trunk. (BORSCHT is an acronym for Battery feed, Overvoltage protection, Ringing, Supervision, Codec, Hybrid, and Testing.) In general, this block of circuitry is different for every type of port circuit.

The heart of the port circuit is the on-board microprocessor, or angel. Every port board in System 75 has an angel that controls the operation of the circuit pack. The angel is implemented as a single-chip microcomputer with up to 8K bytes of firmware. The firmware is divided into two sections: a common portion, which is essentially the same for all circuit packs, and an application portion.

Both the NPE and the SAKI are operated as peripherals to the angel. When setting up a network connection, for example, the following actions occur. The control complex formulates a down-link mes-

sage and sends it over the control channel. The message is received by the SAKI and passed to the angel. The angel sends an acknowledgment (via the SAKI) and examines the message. It then loads the proper time slot and gain values into the NPE so that the desired connection is established.

The distributed intelligence of the angels is a key element which makes a common control channel message set possible. The angels also play an important role in the maintenance strategy. The angel's responsibilities include:

1. Scanning the station/trunk and reporting any state changes uplink to the control complex.
2. Interpreting received (down-link) control channel messages and taking the proper action. For example, when a 'ringer-on' message is received, an analog line-circuit angel must close a relay to provide 90-volt ringing, do ring-cycle timing, etc. A digital line circuit angel, however, would format a command to the station set and send it, using the DCP signaling channel.[2]
3. Handling all short-duration timing functions. Examples include ring cycle and LED cadence timing for stations, outpulsing for trunks, and interdigit timing for DTMF receivers.
4. Performing a variety of maintenance tests. In addition to tests which are run on command from the control complex, the angel does extensive in-line error testing during the normal operation of the circuit pack. Error pegs are kept and reported uplink.

In summary, the angel provides the intelligence necessary to isolate call processing software from port-specific differences and to off-load the control complex from having to do real-time intensive port scanning functions. These are reflected in the virtualization provided in the control channel message set.

This cleanly defined, message-based control interface, coupled with the universal slot concept provides yet another benefit. It is relatively straightforward to design new types of port circuits and integrate them into System 75. Our current family of port circuits includes five types of line circuits (analog, hybrid, Multibutton Electronic Telephone [MET], digital, and data); five types of trunks (central office, direct-inward-dial, tie, auxiliary, and DS-1[7]); and four types of service circuits (tone/clock generator, tone detector, pooled modem, and speech synthesis). Most port boards provide eight port circuits. In the case of the digital line circuit, this results in 16 network appearances, since each port supports two information channels in the digital communications protocol.

Many of the station types are supported across the product family. Analog sets are supported on all AT&T Information Systems PBXs. The hybrid set was adapted from the *Merlin*™ communications system

and is also supported by the *Dimension*® System 85 communication system. The MET set provides an economical migration path for customers who already own *Dimension* or *Horizon*® communications systems. The digital stations provide advanced voice/data features using the digital communications protocol and are common with System 85. This variety of ports and stations allows the system to be tailored to the specific needs of each customer in a cost-effective manner.

2.5 Digital line circuit

To illustrate the concepts previously mentioned, a particular circuit pack, the digital line circuit, is discussed in this section in more detail. Like all System 75 port circuits, the digital line circuit makes extensive use of custom VLSI devices and performs many functions in firmware rather than hardware. The digital line circuit, which terminates eight DCP lines, is an evolutionary step towards an Integrated Services Digital Network (ISDN).[8] Like the proposed ISDN interface, each DCP line provides two information channels and a separate channel for signaling, thereby supporting simultaneous integrated voice/data communication. Thus, this circuit pack supports sixteen endpoints—a density unmatched by any of the other port boards.

2.5.1 Hardware configuration

Figure 7 is a photograph of the digital line circuit. The major functional blocks are indicated on the figure. The five integrated circuits at the upper left are the bus transceivers. The SAKI device, which provides hardware support for the control channel, is at the right of the bus transceivers. Note that the SAKI, like several other devices on the board, is packaged in a 68-pin surface-mount chip carrier. (Physical design considerations are explained in more detail in Ref. 9.) To the right of the SAKI are four NPEs, which provide access to the TDM buses for the 16 information channels that this circuit pack supports. The angel microprocessor that controls the operation of the circuit pack, and its associated RAM are at the right side of the circuit pack. All of the components mentioned above are common to all System 75 port circuits, and are also shown in the generic port board diagram (Fig. 6).

The remaining circuitry (called the BORSCHT in Fig. 6) interfaces directly to the DCP lines. The bottom half of the circuit pack contains eight identical blocks of circuitry. The Digital Line Interface (DLI) device contains a complete 160-kb/s modem packaged in a 40-pin DIP (Dual In-line Package). It provides full-duplex operation over up to 5000 feet of 26-gauge cable, and includes circuitry for framing, scrambling (to reduce radiated noise), clock recovery, and automatic equal-

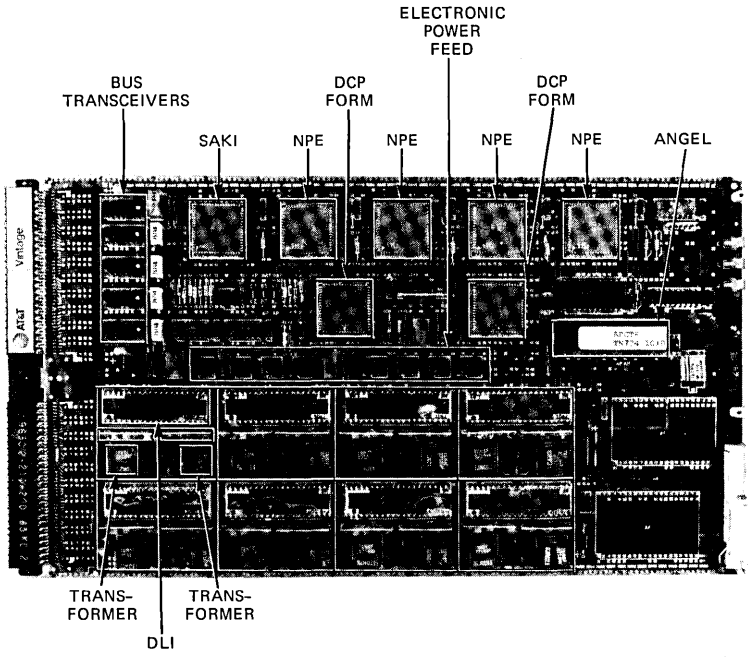


Fig. 7—Photograph of System 75 digital line circuit with major functional blocks indicated.

ization. The two SIPs (Single In-Line Packages) immediately below the DLIs contain the external resistors and capacitors needed by the DLIs. A pair of transformers provides the actual interface to the DCP lines. The use of a transformer-coupled interface has a couple of advantages: it protects the board against longitudinal surges, and it allows power to be supplied to the station over the same pairs of wire (via a technique known as "phantom powering").

The Electronic Power Feed (EPF) chips which are in the middle of the circuit pack control station power. An EPF is a microprocessor-controllable electronic circuit breaker. The EPFs automatically shut down when an overcurrent condition is detected and can also be turned on and off by the angel. In addition, the angel can read the status of each EPF and determine (1) whether it is supplying a normal amount of current to the station, (2) whether it is not supplying any current to the station (this normally means that the station is unplugged), or (3) whether it is in overcurrent mode, which indicates a fault condition in either the station or wiring.

The DCP formatters are custom integrated circuits that provide link-level hardware support for the DCP signaling channel. Since most of the DCP signaling channel protocol is implemented in firmware,

further discussion of the DCP formatters will be deferred until the next section.

2.5.2 Firmware interactions

The digital line circuit angel firmware has three main functions:

1. It processes control channel messages to and from the archangel, as described in Section 2.2.
2. It translates between the control channel (CCMS) protocol and the DCP signaling protocol.
3. It performs a number of maintenance functions, such as logging and reporting transmission errors.

The digital line circuit angel firmware is built around a task dispenser known as APEX (Angel Processor Executive), while real-time I/O to the SAKI and DCP formatters is interrupt driven. Processing DCP messages will be described in more detail, since it is the most complex of the above functions.

The DCP provides an 8-kb/s signaling channel that uses a simplified High-level Data Link Control (HDLC) protocol. In particular, the framing, bit stuffing, Frame Check Sequence (FCS), and link initialization commands (SABM, DM, and UA) are identical to HDLC.[10]

The DCP formatter devices each provide link-level hardware support for four DCP links. In the uplink direction (station to PBX), the formatters provide flag detection, bit de-stuffing, and message demarcation. They generate an angel interrupt when a message byte has been assembled (approximately every millisecond during a message transfer). The angel stores the received bytes in a buffer until the formatter indicates that the complete message has been received.

The completed message is processed at base level in the next APEX task cycle. The angel calculates and verifies the FCS, checks the sequence number, and transmits an acknowledgment—called a Receive Ready (RR)—to the station. The message is then converted to CCMS format, and moved to a different buffer to await uplink transmission over the control channel.

In the downlink direction, messages received over the control channel are converted to DCP format during an APEX task cycle. This includes prepending the correct header and sequence number, and calculating and appending the correct FCS. The message is delivered to the DCP formatters one byte at a time via an interrupt-driven routine. The formatters take care of flag generation and bit stuffing. The angel retains the message in its buffer until an acknowledgment is received from the station. As in HDLC, if none is received within a specified period, the message is retransmitted a maximum of two times. If no acknowledgment is received after the third try, the angel attempts to reinitialize the link.

The two information channels on a DCP link use different logical channels for signaling. Thus, the angel must maintain 16 HDLC-like protocols simultaneously. The angel has responsibility for all link-level functions, including link initialization, sequence numbering, FCS generation and checking, acknowledgments, and retransmissions. As mentioned previously, this allows the call-processing software to maintain a uniform message-based (CCMS) interface to all types of endpoints.

2.6 Digital signal processing technology

Digital signal processing technology is used extensively on all the System 75 service circuits. The AT&T Digital Signal Processor (DSP) integrated circuit[11] is used to implement the signal processing algorithms in System 75. Its advantages include small size, high reliability, low cost, low power consumption, and the availability of numerous development tools.

Some of the many uses of the DSP within System 75 are:

1. The tone/clock circuit pack uses two DSPs to digitally generate all the various tones used by the PBX (e.g., dial tone, ringback, busy tone, intercept tone).
2. The tone detector uses DSPs to implement both Dual-Tone Multifrequency (DTMF) receivers and general-purpose tone detectors (for detecting dial tone, modem answer tone, maintenance tones, etc.) on a single circuit pack.
3. The pooled modem circuit pack contains conversion resources to convert 212A modem signals into DCP format.[2] DSPs are used to implement two 212A-compatible modems on the circuit pack. The advantages of this circuit pack over conventional modem pools include lower cost, uniform administration, better maintenance, and reduction of PBX-room clutter.
4. The speech synthesis circuit pack uses DSPs both for DTMF receivers and for generating Multiple Pulse Linear Predictive Coding (MPLPC)[12] speech samples from stored coefficients.

III. CONTROL COMPLEX

The System 75 control complex is shown in Fig. 1. The control complex is often referred to as the Switch Processing Element (SPE). It consists of a processor, memory, and I/O connected by a single-master Memory Bus (MBus). This configuration meets the cost, performance, and reliability goals for basic service and it can be expanded to support optional services.

3.1 Processor

The processor consists of a commercial 16-bit *Intel** 8086 microprocessor and a Memory Management Unit (MMU) implemented in custom gate arrays. The microprocessor and the MMU functionality were chosen to provide good performance for the largest system configurations and minimum cost for the smallest configurations. Specific constraints are:

1. To minimize equipment cost, the processor and MMU are implemented on a single circuit pack.
2. For maximum performance, most memory accesses are accomplished with only two wait states, including memory management and error correction overhead.
3. Multiple contexts and fast context switching are supported to achieve maximum operating system performance.
4. A high degree of self-checking and protection is provided for call processing applications.

Design trade-offs were made between hardware and software to meet these constraints. The result is an MMU which supports 16-bit virtual to 24-bit physical address mapping, 15 segments of up to 64K bytes each, and the following protection features:

1. Two levels of execution privilege (system and user).
2. Bounds checking on any access, with an overflow stack to aid recovery from stack exceptions.
3. Illegal instruction detection (e.g., HALT instruction).
4. Segment write-protect capability.
5. Distinction between text and stack/data segments to prevent execution of data and to provide execute-only access of text.

3.2 Memory

Because System 75 is software-intensive, the memory can have a significant impact on system cost, reliability, and performance. To meet the system design objectives, the memory uses 256K Dynamic Random Access Memory (DRAM) devices and Error Detection and Correction (EDC) logic. Each memory circuit pack provides 2M bytes organized into 22-bit words (16 data bits + 6 check bits). The EDC circuitry provides single-bit error correction and double-bit error detection and therefore dramatically improves the system's mean time to critical failure. The memory uses VLSI devices to incorporate all refresh, control, and maintenance functions on each pack, thereby eliminating any external memory control function.

---

* Trademark of Intel Corporation.

3.3 Input/output

The I/O functions are implemented with intelligent interfaces which off-load the processor and shield call processing software from real-time-critical tasks. The processor communicates with the interfaces through dual-port memories on the MBus.

3.3.1 Network control

As previously discussed, the network control circuit pack provides the bridge between the control complex and the communication network. It is the master of the TDM bus control channel and, in addition, provides a time-of-day clock with battery holdover, a system clock failure detector, and four switched data channels used for dial-up maintenance/administration and printer output.

3.3.2 Tape interface

The tape interface circuit pack with associated tape drive provides 20M bytes of storage on a 1/4-inch cartridge tape for program load, patches, and translation. The tape drive provides an intelligent memory-mapped interface. It supports both streaming and edit modes and provides extensive error detection and correction capabilities, including the ability to correct very long burst errors.

3.3.3 Maintenance

The maintenance circuit pack uses microprocessors, VLSI, and digital signal processors to provide the following:

1. An RS-232-C interface to a hardwired maintenance/administration terminal (known as the system access terminal), and a low-level user interface in firmware that supplements the high-level interface in software.
2. A tip/ring interface to the remote maintenance center via the central office for automatic alarm reporting. It includes an autodialer, 212A modem emulation, and Level 2 X.25 protocol termination.
3. Cabinet environmental monitoring. If the temperature rises too high, the system is switched to power-fail transfer mode. When the temperature gradient across the cabinet increases to a predefined threshold, the user is reminded (via the system access terminal) to clean the air filters.
4. Power supply and battery holdover monitoring and control. The battery charger and power supplies are constantly monitored and controlled. On ac power failure, the entire system is powered from the batteries for ten seconds. Then the port carrier supplies are shut down and the control carrier is held over an additional ten minutes. Thus,

most commercial power outages are bridged without any interruption in service.

5. Power fail transfer control. As explained above, after ten seconds of battery holdover, the port carriers are shut down. At this time, selected voice terminals are connected directly (via relays) to central office trunks to provide emergency phone service.

The maintenance architecture is described in more detail in Ref. 13.

3.4 Extensions

The control complex can be extended by adding an interface to a multimaster System Bus (SBus), as shown in Fig. 8. In System 75, the SBus supports an additional processor with I/O that terminates the

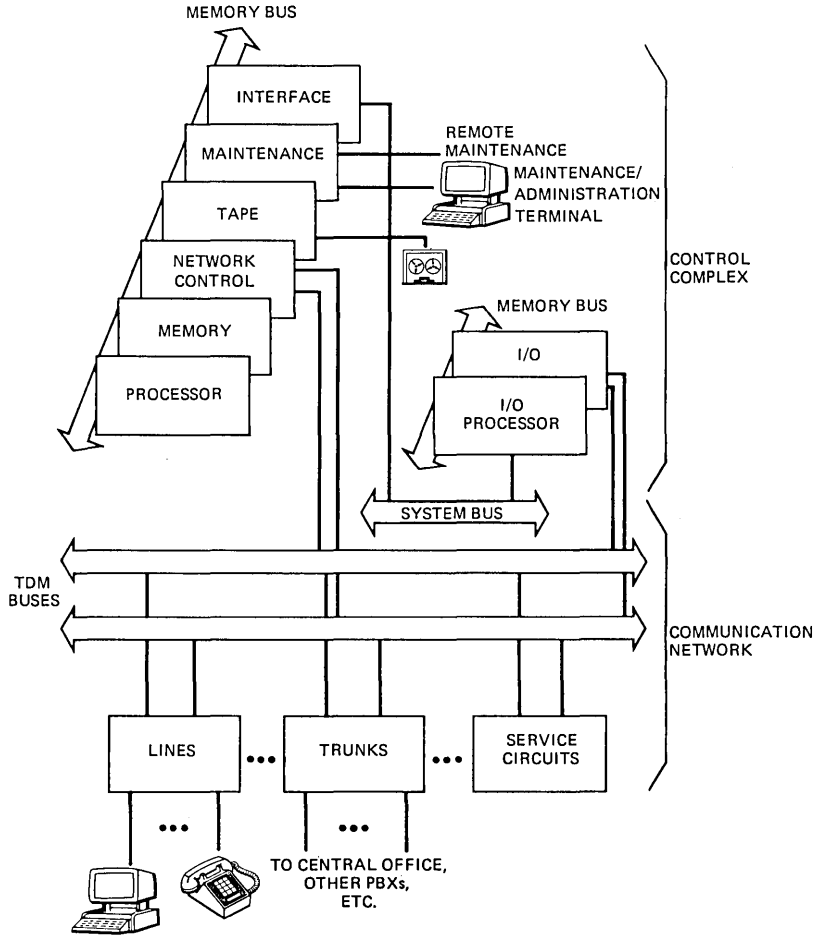


Fig. 8—System 75 communication and control architecture with I/O processor.

switched X.25 channels. These channels connect to adjunct systems such as the applications processor, or other nodes in a Distributed Communications Service (DCS) network.[14]

The I/O processor has its own 16-bit microprocessor plus 128K bytes of RAM and an SBus interface. It connects to the I/O interface through another short MBus. The I/O interface provides additional flexibility because it connects to the TDM buses and terminates the X.25 protocol and the underlying DCP protocol on all four channels. This permits the use of standard data switching, cabling, and termination features for a wide variety of system arrangements.

IV. SUMMARY

System 75 provides a digital communication network that serves the voice and data communication needs of medium-sized customers. Conferencing and a flexible gain plan are integral parts of the network. The control complex efficiently supports a modern operating-system-based software package, and can be expanded to support additional optional features. A wide range of station equipment is supported so that the system can be configured to fill the customer's needs in a cost-effective manner.

To protect the customer's investment, System 75 uses a flexible and highly modular architecture so that the system may be expanded to meet future needs. In addition, the modularity allows economical upgrading of the system as technology progresses.

REFERENCES

1. Bell Telephone Laboratories, *Transmission Systems for Communications*, Fifth edition, 1982, Chapter 28.
2. G. M. Anderson, J. F. Day, and L. A. Spindel, "A Communications Protocol for Integrated Digital Voice and Data Services in the Business Office," Proc. Sixth Int. Conf. on Computer Communication, London, September 1982, pp. 367–72.
3. H. G. Alles, "The Intelligent Communications Switching Network," IEEE Trans. Comm., *COM-27* (July 1979), pp. 1080–7.
4. L. A. Baxter, P. R. Berkowitz, and C. A. Buzzard, "Distributed Digital Conferencing System," U.S. Patent No. 4,389,720, June 21, 1983.
5. B. S. Moffitt and A. R. Ross, "Digital Conference Time Slot Interchanger," U.S. Patent No. 4,382,295, May 3, 1983.
6. B. S. Moffitt and A. R. Ross, "Multiport Memory Array," U.S. Patent No. 4,395,765, July 26, 1983.
7. "Digital Channel Bank Requirements and Objectives," AT&T Technical Reference PUB43801, December 1978.
8. "Integrated Services Digital Networks," Special Issue of IEEE Comm. Magazine, 22, No. 1 (January 1984).
9. A. S. Loverde et al., "System 75: Physical Architecture and Design," AT&T Tech. J., this issue.
10. A. Meijer and P. Peeters, *Computer Network Architectures*, Rockville, MD: Computer Science Press, 1982, Chapter 4.
11. Special issue on "Digital Signal Processor," B.S.T.J., *60*, No. 7, Part 2 (September 1981).
12. B. S. Atal and J. R. Remde, "A New Model of Pulsed LPC Excitation for Producing Natural Sounding Speech at Low Bit Rates," Proc. ICASSP '82, Paris, France, May 1982, pp. 614–17.

13. K. S. Lu, J. D. Price, and T. L. Smith, "System 75: Maintenance Architecture," AT&T Tech. J., this issue.
14. R. S. Divakaruni, G. E. Saltus, and B. R. Savage, "New Directions in Enhanced Voice Networking," Proc. Sixth Int. Conf. on Computer Communication, London, September 1982, pp. 362–6.

AUTHORS

L. A. Baxter, B.S.E.E., 1975, Rochester Institute of Technology; M.S.E.E., 1977, University of Delaware; Bell Laboratories, 1977–1982; AT&T Information Systems Laboratories, 1983—. Mr. Baxter was initially involved with the exploratory design of office communication systems. Since 1980 he has worked on the design of the System 75 communications network. He currently is Supervisor of the System 75 Digital Switching Hardware group. Member, IEEE, Tau Beta Pi, Phi Kappa Phi.

P. R. Berkowitz, B.S.E.E., 1974, Columbia University; M.S.E.E., 1975, Columbia University; Bell Laboratories, 1975–1982; AT&T Information Systems Laboratories, 1983—. Mr. Berkowitz is Supervisor of the System 75 Circuit Design group. He was previously a member of the design team for the *Horizon*® communications system.

C. Alan Buzzard, B.S.E.E., 1964, M.S.E.E., 1965, Cornell University; Bell Laboratories, 1964–1982; AT&T Information Systems Laboratories, 1982–1983. Present affiliation Bell Communications Research, Inc. Mr. Buzzard's past responsibilities have included development of modems, data networks, and voice/data PBXs. His present interests are in speech coding, speech synthesis, and automatic speech recognition. Member, IEEE.

J. J. Horenkamp, B.S.E.E., 1964, St. Louis University; M.S.E.E., 1966, Columbia University; Doctor of Engineering Science, 1973, Columbia University; Bell Laboratories, 1964–1982; AT&T Information Systems Laboratories, 1983—. Mr. Horenkamp is Head of the System Design department responsible for hardware and firmware development and maintenance planning of System 75. He has been involved with exploratory development and final design of a variety of PBXs and customer premises telecommunication systems.

Frank E. Wyatt, B.S.E.E., 1969, M.S.E.E., 1970, University of Illinois; Bell Laboratories, 1971–1982; AT&T Information Systems Laboratories, 1983—. Mr. Wyatt has worked on a variety of business communication and management information systems. Since 1980 he has supervised the group that developed the control complex for System 75. Member, IEEE.

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

Physical Architecture and Design

By A. S. LOVERDE, H. D. FRISCH, C. R. LINDEMULDER, and D. BAKER*

(Manuscript received July 11, 1984)

This paper discusses the physical architecture, the rationale for design choices, and the physical design of the System 75 office communication system. The design features a single equipment cabinet housing up to 720 ports, a display-enhanced attendant console, and a modular-jack-based station-administration facility. The architecture minimizes the small system cost while providing modular building blocks for feature additions and growth. Customer participation in maintenance and administration is encouraged by attention to human factors in design and labeling details.

I. OVERVIEW OF PHYSICAL ARCHITECTURE

The main goals of the physical architecture of the System 75 office communication system are to maximize the amount of service that can be provided by a single-cabinet system, to provide a modular, cost-effective design over a broad range of sizes and needs, and to provide an aesthetically pleasing functional design that will enhance customer participation in system maintenance and administration.

A single cabinet (Fig. 1) houses all equipment needed to support up to 720 ports with any mix of station types or trunks. The common

* Authors are employees of AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc.

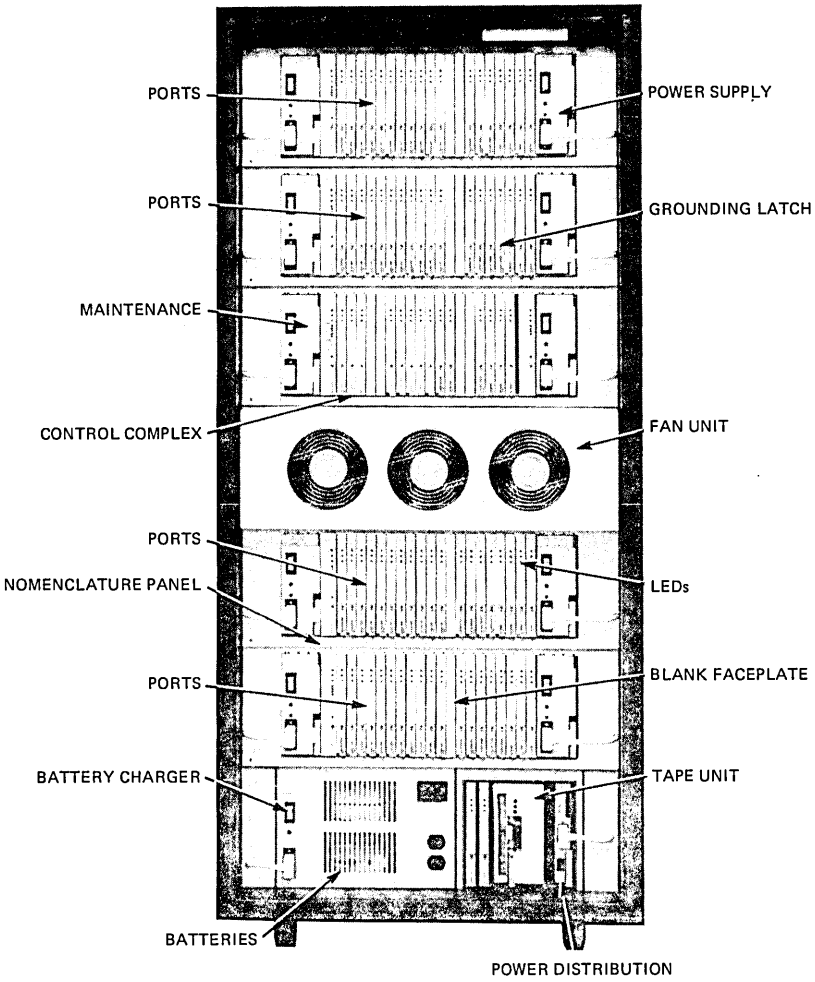


Fig. 1—Equipment cabinet.

equipment complement including the control complex (processor, memory, tape interface, and maintenance circuit packs), fan unit, tape unit, battery charger, and other power-distribution equipment occupies only a small fraction of the cabinet; this leaves ample room for the addition of port carriers, power units, and port circuits.

A new modular-jack-based station-administration facility (Fig. 2) and a new attendant console (Fig. 3) enhance system flexibility and complement the powerful software-based administration and maintenance features.[1,2]

II. CABINET-LEVEL DESIGN

A key trade-off in PBX design is balancing the amount of common

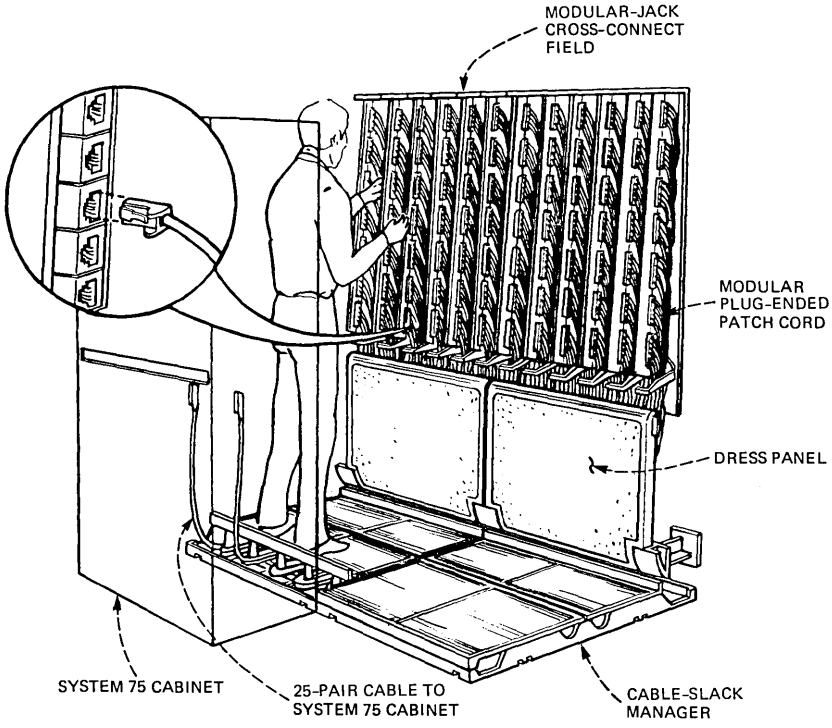
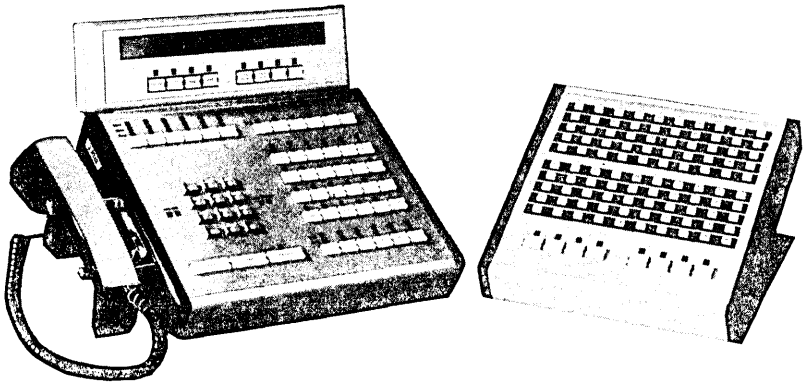


Fig. 2—Station administration facility.

equipment needed to get started (intercept cost) with the amount of modular equipment required to meet a particular set of customer needs (slope). In addition, the designer must carefully balance the costs of ordering, stocking, maintaining, and administering the system with the cost of the hardware elements themselves. Since the System 75 market spans a large range of sizes and features, minimizing the impact of this trade-off was a major challenge at each step in the design process.

To minimize ordering complexity, simplify installation, and maximize production volume, all common equipment was designed to function over the full range of system configurations. To this end, all wiring required to support the full complement of port carriers is provided in a connectorized manner in all cabinets. The cooling fans are designed to accommodate the maximum equipment load under worst-case conditions. The automatic monitoring and maintenance elements likewise are sized to accommodate any mix of equipment, while detailed configurations are specified in the translation software. Power cabling and battery capacity are similarly designed for worst-case maximum load. With this strategy, the special engineering of



(a) (b)

Fig. 3—(a) Basic service attendant position console. (b) Direct extension selection console.

power and cooling normally associated with a PBX of System 75's complexity is virtually eliminated. Cost of the hardware elements is also minimized by eliminating factory handling of multiple design options and special orders.

2.1 Cabinet

A new welded steel cabinet that uses side panels as structural members was developed. This monolithic design eliminated nearly 32 feet of EMI gasketing as well as the expense of handling removable side panels. A new structural foam door assembly complements the new cabinet. Both cabinet and door have a tough, durable, textured finish, which masks surface imperfections and scratches. A simple pin hinge mechanism with a detent permits the door to stay in an open position for ease of maintenance and allows multiple cabinet lineups where the system is installed together with other product family members. EMI integrity is provided by a gasket contacting conductive paint on the cabinet body. The rear is EMI-sealed by inexpensive metal plates screwed directly to the cabinet frame.

The full-size System 75 cabinet is 32 inches wide, 24 inches deep, and 70 inches high. Fully equipped it weighs approximately 800 pounds. To accommodate smaller customer needs, a 42-inch-high cabinet was designed for up to 240 ports (Fig. 4).

2.2 Equipment cooling

Customer premises systems encounter a wide variety of operating environments. In addition, the configuration flexibility of System 75

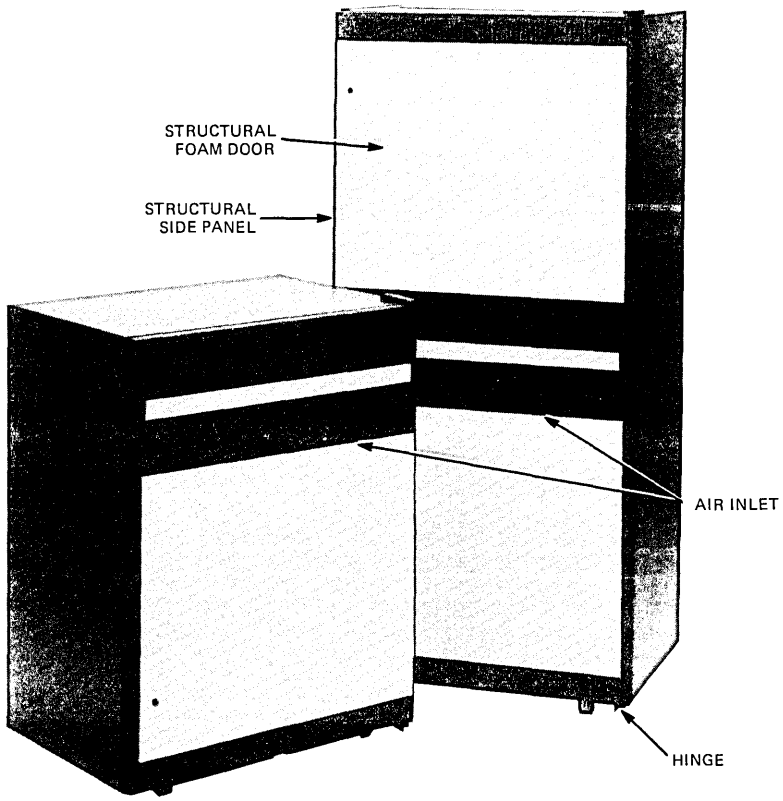
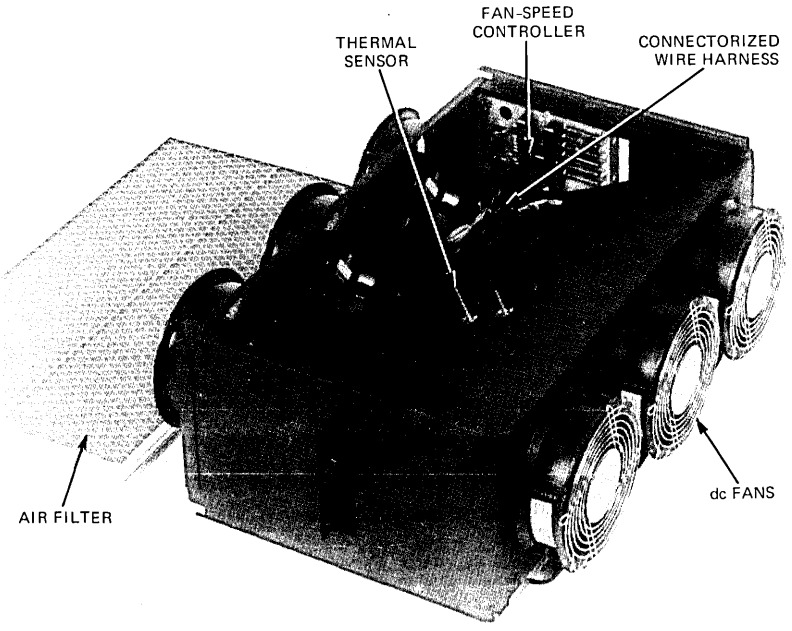


Fig. 4—Small and medium cabinet configurations.

presents a wide range of thermal loads. In keeping with the common equipment philosophy, the fan unit was designed to cover the maximum anticipated load of 2500 watts under the worst-case design criteria of 120°F at 10,000-foot elevation. Early analysis confirmed by laboratory measurements indicated that a three-fan unit using available dc fans operating at 48V would fit within the height of a carrier module and could effectively cool up to three fully loaded carriers. By placing two such units (six fans) in a central location, one directed upward and the other downward, a full cabinet could be cooled.

Additionally, the thermal design combined alarms for high-temperature fan failure, clogged filters, and other maintenance needs together with a fan-speed controller to adjust fan speeds automatically to compensate for varying thermal loads and operating environments. The resultant design (Fig. 5) is a 9-inch-high integrated module housing all six fans; removable, washable air filters; fan-speed con-



troller; and connectorized wiring to mate with power and alarm leads in the cabinet wiring harness.

Secondary design benefits include cleaner inlet air than is available at the base of the cabinet, low acoustic fan noise when full fan power is not required, and a separately testable unit that can be quickly replaced in the field.

The small cabinet uses the same design with three fewer fans at reduced cost.

2.3 Cabinet wiring

The system architecture includes a Time Division Multiplexed (TDM)* bus and distributed power, which allows simple internal cabinet wiring. The TDM bus is terminated at each end by a paddle-board-mounted bus terminator and is interconnected from carrier to carrier using inexpensive flat cables terminated on paddle boards that plug directly onto the backplane pins.

Field installation of additional carriers is accomplished through the addition of a flat cable and reuse of the bus terminator. The remaining

*Acronyms and abbreviations used in the text are defined at the back of the *Journal.*

intercarrier wiring consists of leads for power unit status and carrier identification. These terminate on a single connector at each carrier position, again allowing for simple replacement or additions in factory or field. The power unit ac/dc input is wired to the front of each unit to permit addition or replacement of modular power supplies without removal of back panels or carriers. All cabinet wiring, independent of size or configuration, is contained in a single connectorized cable harness designed to simplify factory assembly.

2.4 Power distribution and tape unit

System 75 is equipped with a high-capacity tape recorder to back up the volatile memory. This unit, along with the standard system battery plant and all required power distribution, filtering, and power factor correction hardware, is housed in a modular unit at the bottom of each cabinet. This unit occupies the vertical space of a single circuit pack carrier (9 inches high) and supports the entire range of system configurations.

III. POWER ARCHITECTURE

In keeping with the overall cost optimization strategy, the power is divided into common elements (battery, power factor inductors, EMI filters, main circuit breakers, etc.) and modular, carrier mounted, switching type rectifiers.

The modular rectifiers use a five-pin connector designed to accept either commercial quality 110 Vac or 144 Vdc from the battery back-up unit as inputs. Regulated outputs of −48, −5 and +5V are provided directly to power buses on the carrier backplanes. Additional voltages or regulation are supplied on the circuit packs, thus creating a standard circuit pack to backplane power interface. In addition, the attendant console and the cooling fans are powered from a modular rectifier. As a result, the entire system requires only one standard 110V, 50A outlet for installation and a single, self-contained battery plant for backup.

This modular architecture readily accommodates international applications since pin-compatible rectifiers that accept different input voltages and frequencies can readily be designed. Similarly, any back-up system providing 110 Vac or 144 Vdc can be accommodated.

3.1 Maintenance and recovery

The use of individual power supplies to power each carrier ensures that the failure group associated with any given supply is minimized. In addition, the power units contain software-resettable circuit breakers. This combination, when coupled with software-driven maintenance, virtually eliminates the need for craft or customer intervention

to replace fuses or reset circuit breakers tripped by an accidental overload or transient conditions. Fusing is used only to meet Underwriters Laboratories requirements and to protect against fire or personnel safety in the event of a catastrophic failure such as damage to the insulation on the power cord.

Since the individual rectifiers are only three inches wide and nine inches high and weigh roughly nine pounds, additions or replacements are easily accommodated.

3.2 Battery holdover

For protection against transient power outages such as those frequently encountered during lightning storms, System 75 uses a 144 Vdc, 2.5 ampere-hour battery plant. This dc power together with the ac line power is distributed by five conductor cables to all switching power supplies within the cabinet.

As described by Lu et al.,[2] holdover strategy is under system software control to ensure optimal use of the available battery power.

3.3 UL and CSA qualification

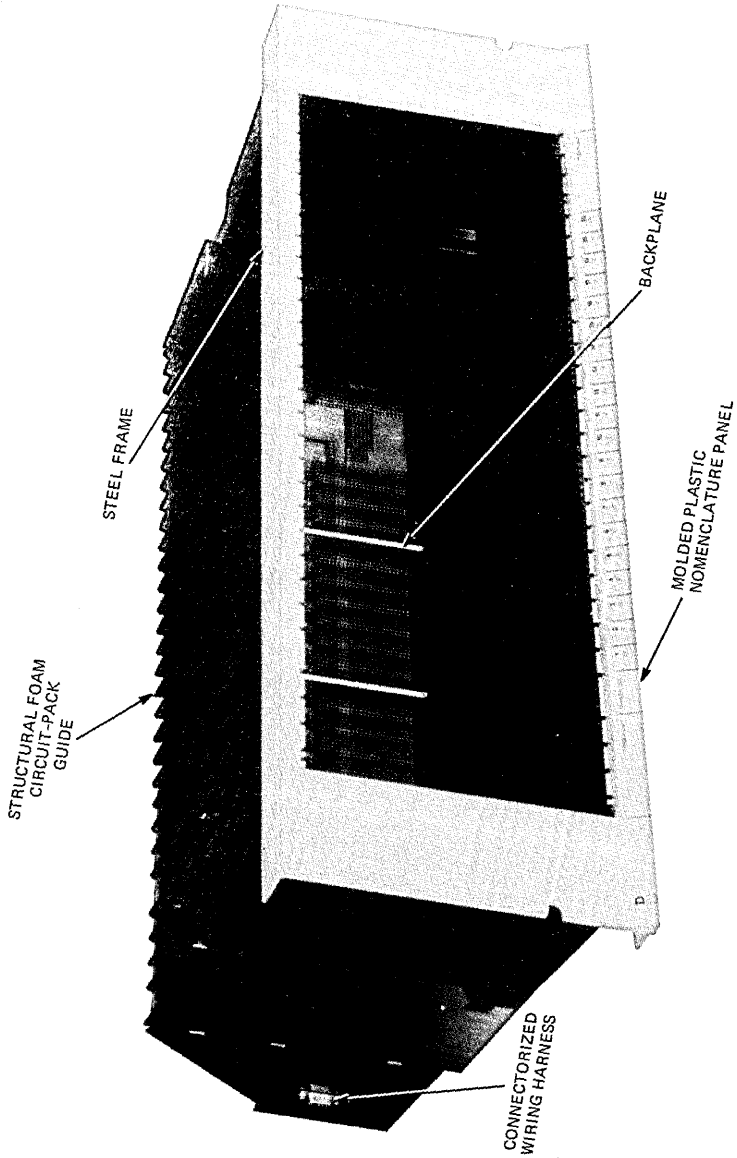
As System 75 will be competing for sales in many new markets, all elements of the system were listed with both Underwriters Laboratories (USA) and the Canadian Standards Association (Canada).

IV. CARRIER DESIGN

A basic building block for System 75 is the newly designed circuit pack carrier shown in Fig. 6. A single structural foam plastic molded part forms the carrier guides and backplane support. The draft angle of the guides, used to allow proper removal of the part from the mold, is also used to introduce a slight bow to the guides when the foam part is attached to the backplane. This inward bow gives just enough pressure to the circuit pack to provide comfortable tactile feedback as the circuit pack is inserted into the carrier.

A formed steel frame is used to provide the structural integrity needed to support a combined circuit pack load of up to 60 pounds. The frame also provides the alignment of the circuit pack in the lower rail of the slot as well as the rigid surface required to accommodate the forces generated by the latch as the circuit pack connector engages and disengages the backplane pins.

A molded plastic part is placed over the steel frame when the carrier is installed in the equipment cabinet to facilitate the alignment of the circuit pack in the upper rail of the slot, to provide a slanted surface for labeling of the carrier, and most importantly, to give a clean, aesthetically pleasing visual surface to the assembled unit.


STEEL FRAME
BACKPLANE
MOLDED PLASTIC NOMENCLATURE PANEL
STRUCTURAL FOAM CIRCUIT-PACK GUIDE
CONNECTORIZED WIRING HARNESS

Unique carrier codes are created by associating different backplanes with the generic building blocks described above.

A single connector brings the carrier identification and alarm leads from the wiring harness to the carrier.

Each port slot is connected to the building wiring by a 25-pair connector terminated directly on the backplane. Through the implementation of the AT&T Information Systems Architecture (ISA), pin assignments for terminals, cross-connect fields, system cables, backplanes, and circuit packs have been coordinated to ensure compatibility with all present and future AT&T Information Systems products. This standard interface allows any of the many unique codes of port circuit to be installed in any port slot. This universal port-slot concept is of considerable value in both the factory and the field.

For System 75, two codes of backplane are required: one providing the dedicated slots for the common equipment along with 10 port slots and a second code housing 20 port slots. Up to four of these port carriers plus one common equipment carrier may be installed in a full cabinet. Field or factory installation simply requires mounting the carrier (four screws), connecting the already provided wiring harness connector, and snapping the molded frame in place. A molded blank is provided where carriers are not required to contain cooling air and to enhance the appearance.

V. CIRCUIT PACKS

5.1 Common design parameters

All circuit packs use the 8-inch by 13-inch *Fastech*™ board outline and the 200-pin *Fastech* connector. Faceplates are sized to fill the width of the slot (typically 3/4 inch) and present a standard pattern of three Light-Emitting Diodes (LEDs) (red, green, and yellow) for uniform maintenance. A special grounding latch protects the circuits from electrostatic discharge during installation.

Manufacturing considerations—such as orienting components for machine insertability, providing lands for automatic testing, and complying with the requirements associated with aqueous cleaning of water soluble flux—also apply to all designs.

5.2 Control circuit design

Control circuit packs include processor, memory, network control, tape control, protocol interfaces, and maintenance. These circuits are unique functional entities occupying dedicated slots in the control carrier and, thus, do not conform to the universal slot concept. High functional density and critical design parameters dictate the use of multilayer board technology. Since each circuit type represents a unique design, details will not be presented.

5.3 Service and port circuit design

Service circuits (pooled modems, tone generator, and tone detector), as well as port circuits (lines and trunks), are designed to meet the universal slot interface criteria. This criteria ensures that unique customer configurations can easily be accommodated, but imposes constraints on the circuit pack designer since signal, power, and ground leads must be uniform over a wide variety of functional designs.

Circuit density is another important parameter on System 75 high-volume line and trunk circuits. Achieving this density required extensive use of surface mounted VLSI, interconnection of components on Hybrid Integrated Circuits (HICs) used both in Dual In-line Package (DIP) and Single In-line Package (SIP) configurations, and the use of multilayer board blanks.

The final design consideration for all port and service circuits is to take advantage of the basic hardware architecture. To this end, all port circuits contain three sections: a system bus interface section that ties the circuit to the time division system bus, a port control processor (angel) section that interfaces this port to the main processor complex, and a functional section that provides the unique interface required for that particular circuit pack's function. Some broad design considerations for each of these sections is given below. A typical 8-port circuit pack is shown in Fig. 7.

5.3.1 System bus interface section

The system bus interface section of each pack, located adjacent to the backplane connector, consists of five custom-designed bus buffers

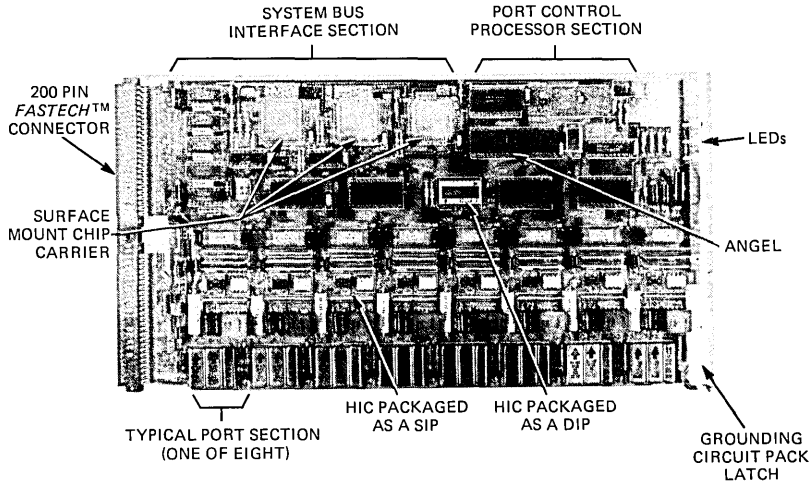


Fig. 7—Central office trunk circuit pack.

that interface the circuit pack to the switch. These buffers are then coupled to custom VLSI devices designed to interface the bus. The devices are located in the same position on each circuit pack to eliminate noise coupling with sensitive analog voice-path circuitry on adjacent circuit packs and to ensure consistent performance.

5.3.2 Port control processor section

Each port circuit pack contains a port control processor (angel) complex, which adapts the system bus interface section to the station interface section. This section, based on the 8051–8052 microprocessor family, performs all required functions to service both the system bus and the station interface, as well as perform automatic maintenance functions.

5.3.3 Functional interface section

Each pack also contains a station, or trunk, or service interface section that provides the interface to an individual type of terminal, trunk, or service. These functional interface configurations are replicated on a port by port basis on each pack to ensure consistent, quiet performance. Designing in this manner allows increased efficiency in noise-reduction efforts, since analysis done on a single port can then be applied to all others. To further reduce audible noise—especially in view of the fact that all circuit packs interfacing analog facilities are designed using double-sided circuit boards—all critical circuit paths on the packs are surrounded by substantial amounts of printed grounding for noise suppression.

VI. ATTENDANT CONSOLE

The attendant console is the most visible and most active station of a PBX. Its design must combine human factors, rugged functional design, and asthetics with efficient, cost-effective manufacture and ease of maintenance.

The functions and operations of the System 75 attendant console are similar to the highly successful *Dimension*® PBX and *Dimension* System 85 consoles to preserve product family architecture and minimize attendant training. Other features such as providing both tactile and audible feedback on button pushes, right- or left-handed handset/headset mounting, and individual tone controls have also been provided in the new design.

Important new functionality includes the use of the standard ISA four-pair wiring plan and Digital Communications Protocol (DCP) to interface the console to the system. This feature allows the attendant console to plug into any modular jack since the console interface to the switch is identical to any other digital station.

An additional new feature is the incorporation of a 40-character alphanumeric display, which, when coupled to the software-based functionality, both facilitates and enhances message retrieval, calling/called party identification, and other critical attendant functions.

6.1 Console physical design

An exploded view of the attendant console (Fig. 8) shows the details

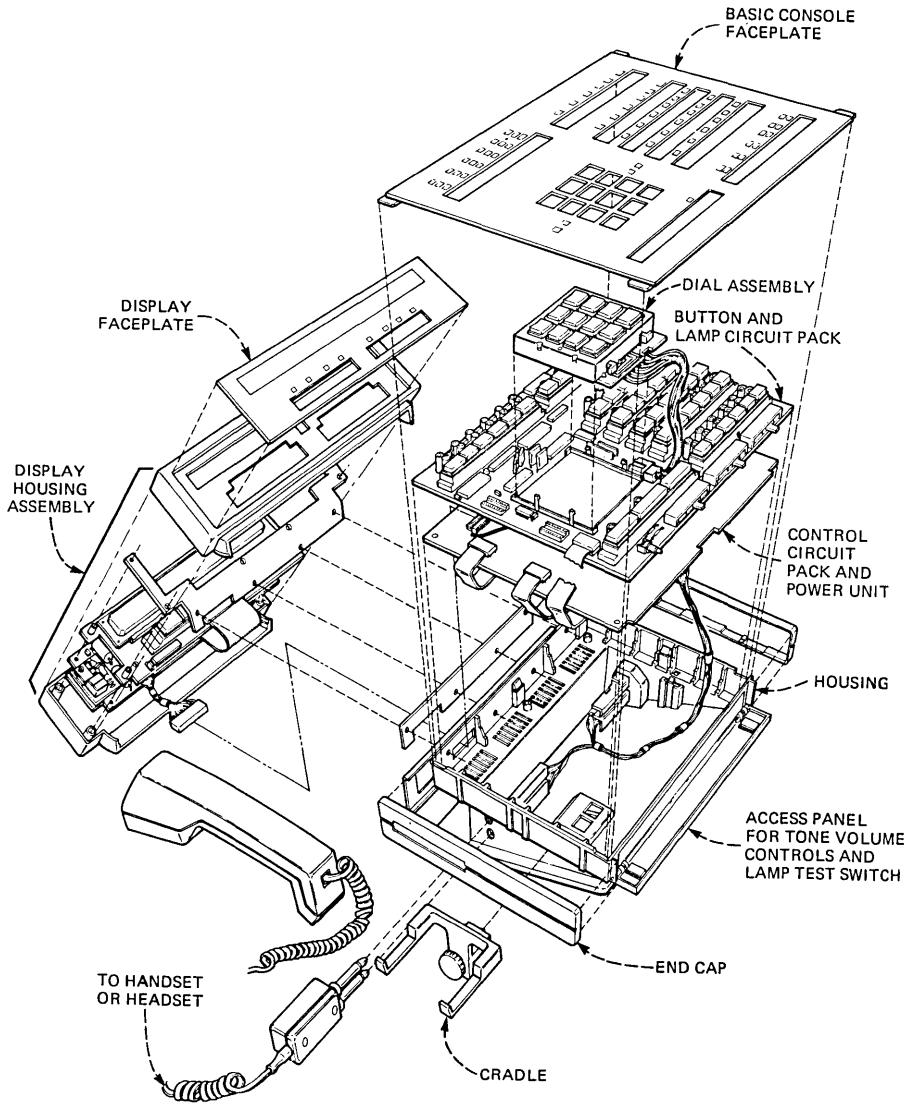


Fig. 8—Attendant console, exploded view.

of the console physical architecture. All components, keys, and LEDs are mounted on independently testable and replaceable units for ease of assembly and repair. The entire unit snaps together for easy assembly. Product family identity with other System 75 terminals is preserved. All interconnection cables are keyed to prevent inadvertent damage. A heavy rubber-footed steel base keeps the console in place as keys are jabbed by the user.

VII. FACILITIES INTERFACE

The main function of any PBX is to connect various facilities (lines, trunks, terminals, host computers, etc.) to one another. A secondary need is to connect auxiliary equipment (music sources, Station Message Detail Recording [SMDR] recorders, etc.), the system administration terminal, and remote maintenance centers to the switch. The range of System 75 interconnection is shown in Fig. 9.

To accommodate this wide spectrum of needs System 75 conforms to the AT&T Information Systems architecture and the four-pair uniform station/terminal wiring plan. This conformance ensures that building wiring, terminals, and most adjuncts installed for use with other AT&T Information Systems products can be used or reused directly with System 75.

The facility interface can further be divided into three areas: the interface between the switch cabinet and its adjuncts, the interface between the stations and the switch, and the interface between the external Central Office (CO) facilities and the switch.

7.1 Switch cabinet interface

Figure 10 shows the external interfaces of the System 75 cabinet. As can readily be seen, the universal port concept and the hardware architecture greatly simplify this interface.

7.2 Station interface

The most important customer need in administration is to perform station moves, upgrades, or installation of additional stations. Complementing the software-driven user-friendly administration terminal is a modular-jack-based cross-connect field (Figs. 2 and 11) human-engineered to accommodate the user with minimal training. Key elements include a cable-slack manager to organize the 25-pair port cables, modular molded plastic cross-connect modules, and modular plug-ended patch cords.

7.2.1 Hardware

The basic hardware building block is a two-piece, hinged, molded plastic column housing small circuit modules containing 25-pair cable

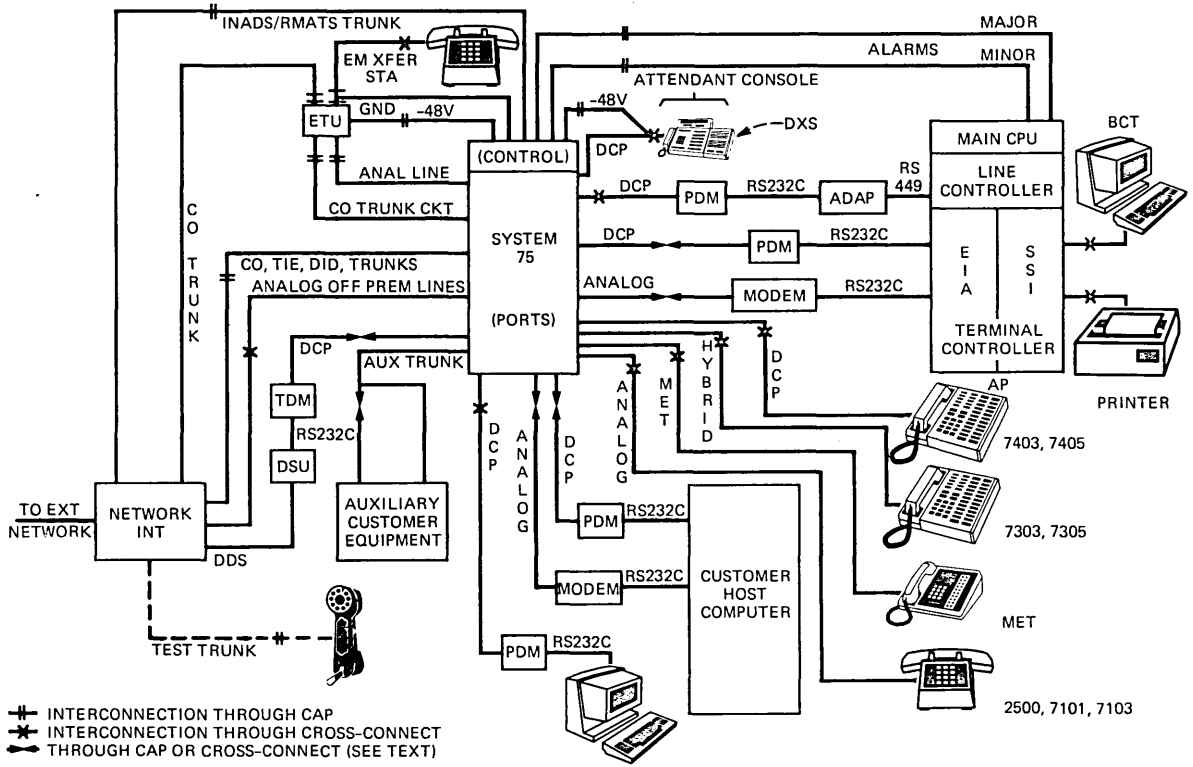


Fig. 9—Facilities interconnection.

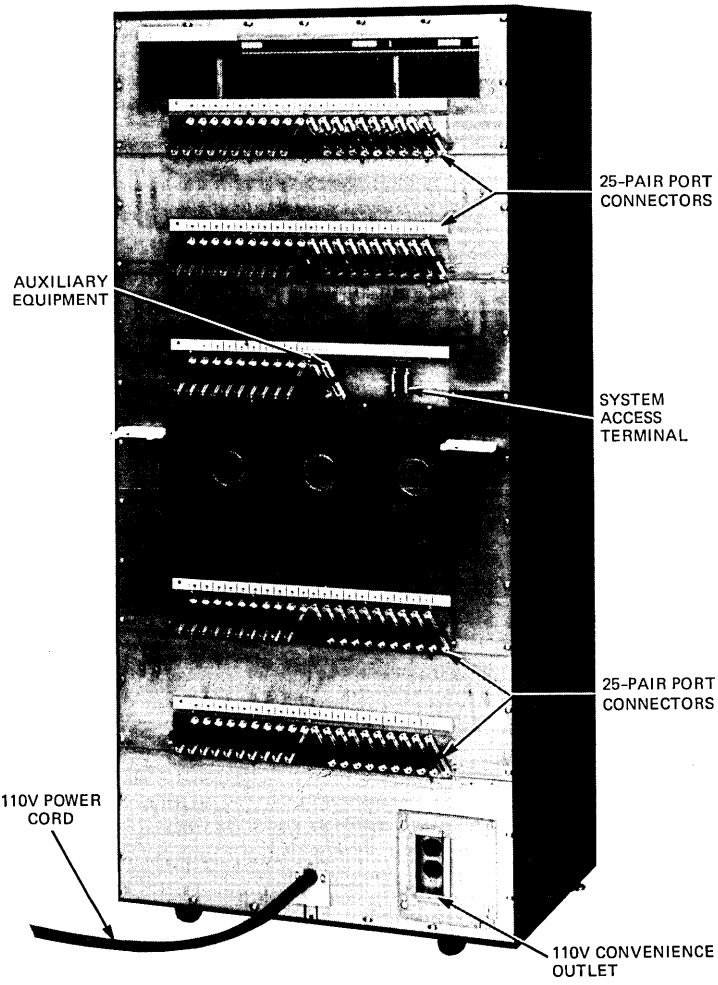


Fig. 10—Equipment cabinet external interfaces.

connectors together with modular jacks. Port circuit modules expand a 25-pair port cable into eight ports consistent with System 75 circuit-pack density, and station modules expand a 25-pair cable into six jacks consistent with the Information Systems Architecture four-pair station wiring plan. Each column accommodates three port-circuit modules and four building-cable modules for a total of 24 stations. Columns attach with interlocking tabs; thus, only the first unit need be accurately installed and leveled. All connecting cables in a column are accessible via a vertically hinged section specially designed to reduce any stresses on the cables induced by opening and closing the unit. The modular patch cords used for the connections are modified by the

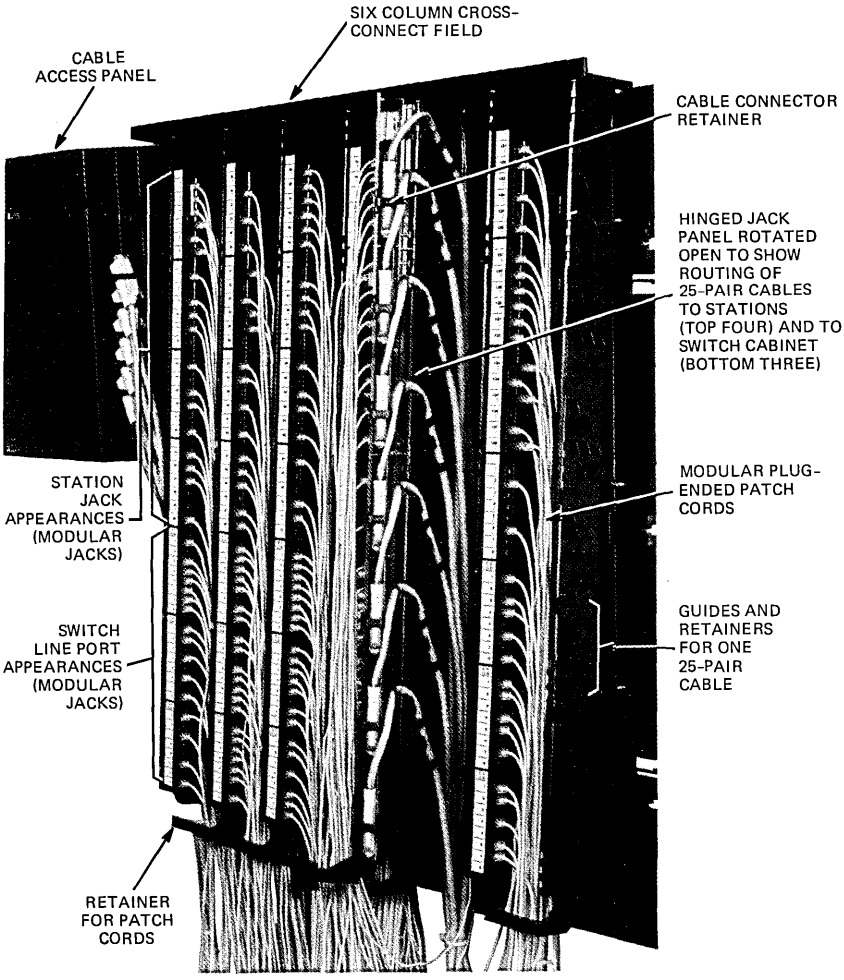


Fig. 11—Modular-jack-based cross-connect field.

addition of triangular wings intended to protect the tab on the modular plug and serve as an anti-snag feature when removing cords.

7.2.2 Labeling

To further facilitate customer participation, a labeling plan employing graphic symbols to represent items such as jacks, carriers, and circuit packs is used. Identical designations (Figs. 12 and 13) appear on the cabinet, carriers, cables, cross-connect, and station wall jacks making rapid station moves or upgrades possible. In addition, the traditional blue and purple PBX colors are used to denote station-side and switch-side connections, respectively.

SYMBOL	FUNCTION	DESIGNATION SEQUENCE
	CABINET	1, 2, 3, . . ., n
	CARRIER	A,B,C,D,E
	SLOT	1, 2, 3, . . . , 20
	JACK	1, 2, 3, . . ., 400
	SITE	A, B, C . . .
	FLOOR	
	BUILDING	

Fig. 12—Symbolic nomenclature system.

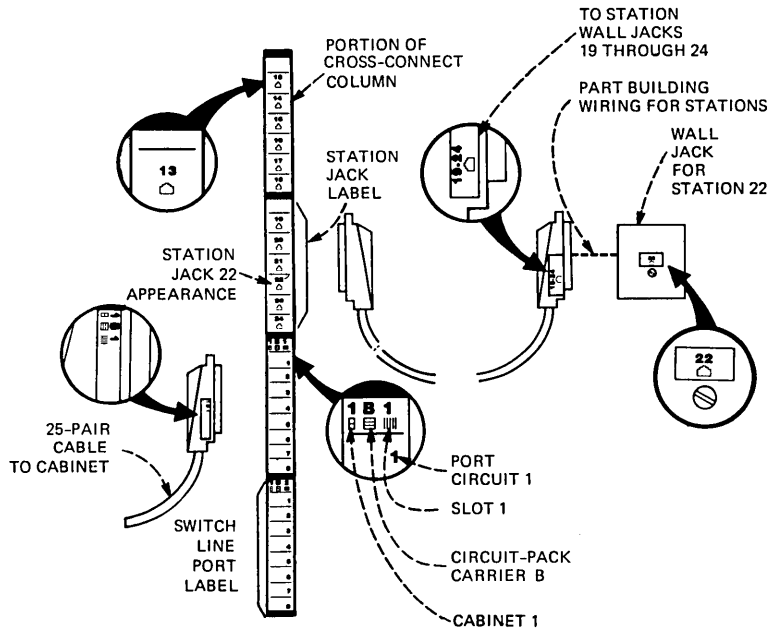


Fig. 13—Equipment labels.

The labeling plan was devised with several human factors considerations in mind. Combinations of letters and numbers are used to identify items since alphanumeric sequences are easier to remember than strings of numbers. Graphic symbols were chosen to represent

items to remove all language association and make the labeling plan suitable for international use.

7.3 External facilities

Consistent with FCC regulations and the uniform building wiring plan, all external facilities present 25-pair connectorized appearances. In principle, these could be connected directly to the switch. In practice, it is essential to fan out these interconnections to provide maintenance and test personnel access to the individual pairs. Since this activity is generally performed by a trained technician and since these connections are seldom rearranged once installed, traditional insulation displacement cut-down blocks are used. In System 75, these blocks along with emergency transfer relays are housed in a factory wired module housed in an inexpensive structural foam housing. This easy-to-install cable access panel is shown in Fig. 14. Trunk rearrange-

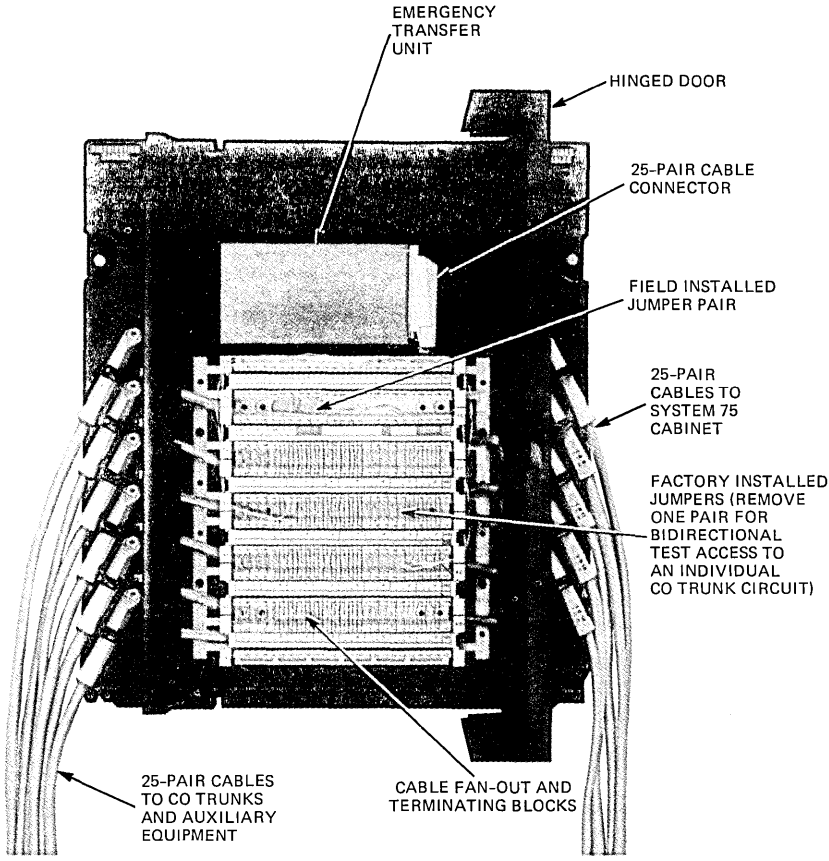


Fig. 14—Cable access panel.

ment and special circuit termination is accomplished by removing straps and wiring directly to the blocks.

Auxiliary customer equipment (music sources for music on hold, paging amplifiers, etc.) is housed in a separate cabinet. Interconnection to System 75 is either by a dedicated cable or by passing through the interconnect hardware.

7.4 Alternate arrangements

While facility-interconnection arrangements have been designed to optimize System 75 installations, the uniform wiring plan assures that System 75 can be connected to existing wiring or to the hardware recommended for larger *Dimension* system 85 installations should such facilities already be in place and/or the customer have such a preference.

VIII. SUMMARY

The goals of maximizing the amount of service housed in a single cabinet while providing a modular physical architecture that can be tailored to the customer's size and feature needs have been met. The software-based administration and maintenance features are effectively complimented by user-friendly hardware ranging from the modular-jack-based station cross-connect field to the simple modular power units. New technology such as surface-mounted VLSI, structural foam carriers, and modular switching regulators is combined with familiar hardware such as modular jacks to provide a user-friendly, cost-effective, manufacturable design.

REFERENCES

1. H. K. Woodland, G. A. Reisner, and A. S. Melamed, "System 75: System Management," AT&T Tech. J., this issue.
2. K. S. Lu, J. D. Price, and T. L. Smith, "System 75: Maintenance Architecture," AT&T Tech. J., this issue.

AUTHORS

Donn Baker, M.E. (Engineering), 1953, Stevens Institute of Technology; Bell Laboratories Communications Development Training Program, 1956; Bell Laboratories 1953–1982; AT&T Information Systems Laboratories, 1983—. From 1953 to 1961, Mr. Baker worked on key telephone system development including circuit design, feature definition, logic design, and system integration of exploratory electronic key systems. From 1961 to 1965 he supervised system, circuit, and physical design of electronic PBXs. During 1965 to 1968, he was an author and editor of the four volume text: *Physical Design of Electronic Systems* and was an instructor for parts of a four semester in-house course on this subject. Since 1968 he has supervised groups responsible for physical, circuit, and software design of a variety of business customer systems. He is currently supervising the System 75 Physical System Arrange-

ments group, which includes design of cross-connect subsystem. Member, IEEE; Professional Engineer, New Jersey.

Howard D. Frisch, B.S.M.E., M.M.E., M.B.A. (Finance/Operations), Cornell University, in 1979, 1980, and 1981, respectively; Bell Laboratories, 1981–1983; AT&T Information Systems Laboratories, 1983—. At both Bell Laboratories and AT&T Information Systems Laboratories, Mr. Frisch has contributed to the system physical design and system architecture of AT&T System 75 and associated hardware.

C. R. Lindemulder, B.S.M.E., 1960, New Mexico State University; M.M.E., 1963, New York University; Bell Laboratories, 1960–1982; AT&T Information Systems Laboratories, 1983—. Mr. Lindemulder began his career with Bell Laboratories as a mechanical designer on the Nike-Zeus and Sentinel radar systems. He was promoted to Supervisor and worked on the development of the Safeguard missile site radar. In 1975 he was transferred to Holmdel to work on PBX development. Past assignments include responsibility for the physical design of the *Horizon*® communication system. He is currently responsible for the physical design architecture and development of the switch portion of System 75.

Albert S. Loverde, B.S. (Mechanical Engineering), 1961, Purdue University; M. S. (Engineering Mechanics), 1963, New York University; Bell Laboratories Communications Development Training Program, 1964; Bell Laboratories, 1961–1982; AT&T Information Systems Laboratories, 1983—. From 1961 to 1971, Mr. Loverde worked on military system development, including missile guidance, radar, nuclear weapons effects, and system integration and test. From 1971 to 1977, he supervised the design of digital transmission systems for use on paired cable, coaxial cable, and optical fiber. From 1977 to 1983, he was the head of the Customer Premises Physical Design department, which developed hardware for a variety of systems, including *Horizon*® and System 75. He is currently Head of the Engineering Information and Standards department. Member, Pi Tau Sigma, Tau Beta Pi.

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

Switch Services Software

By W. DENSMORE, R. J. JAKUBEK, M. J. MIRACLE, and J. H. SUN*

(Manuscript received July 11, 1984)

The switch services software of System 75 provides the basis for an extensible office communication system, supporting a wide variety of voice and data-switching services. This paper presents the software architecture of the System 75 switch services. The concepts of user, group, and process-per-call form its foundation. We introduce the architecture by stepping through a simple station-to-station phone call, and proceed to the derivation of a call model based on the topology of a call. This call model is realized as a layered set of cooperating processes that execute under the Oryx/Pecos Operating System on the System 75 switch processor. The software layers and processes are discussed and a call walk-through is used to illustrate the process interactions.

I. INTRODUCTION

The System 75 software supports a wide spectrum of terminals that range from a single line station to a sophisticated digital station for simultaneous voice and data communications. It also supports more than 150 features for office and business communication needs. Among them are the station features for handling multiple, simultaneous calls, features used for covering unanswered calls, routing features to select

* All authors are members of AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc.

the least expensive network facility, capabilities that allow messages to be left for the called party automatically, and terminal dialing and modem pooling features for data communications services.

This paper describes the switch services software architecture of System 75, a framework that supports a wide variety of features and terminals. Section II states the design challenges and Section III illustrates a basic call scenario leading to the derivation of an essential call model. The concepts of user, group, and call are introduced, and are mapped to a basic software structure. In Section IV, this structure is generalized to a layered software architecture consisting of a set of cooperating processes. A call walk-through is used to illustrate the interactions among processes.

II. DESIGN CHALLENGES

The challenges in building switching software for an office communication system include:

- The vast number and variety of features to be supported
- The need to integrate different office services
- A wide variation in the capabilities of existing and future terminals
- Stringent real-time response criteria
- The asynchronous and concurrent nature of the external world.

Our design began with an analysis of the feature operations and resource management requirements of a switching system. Then, the essence of the feature and terminal operations were extracted into functional modules and the basic primitives of the system were defined. Next, we formulated a call model by analyzing the dynamic behavior of a call and the relationships between the functional modules. The functional modules were then layered into a set of cooperating processes, with the primitives of the system provided through message-based interfaces. Finally, information hiding and synchronization techniques were applied to simplify the software structure and to enforce stronger partitioning of functions.

A primary goal of this modular and disciplined architecture is to minimize the effort required to add new terminals and features. This architecture should also be easy to understand by software developers so that the architectural integrity is preserved over the product life. Moreover, the architecture should simplify the integration of more data-processing-like functions in the future.

Trade-offs exist between implementing the design goals and meeting the real-time requirements of a switching system. For example, a virtual terminal interface provides for uniform implementation of features across all terminal types at the expense of access time to the terminal. The primitives of the system are carefully defined to balance between generality and efficiency.

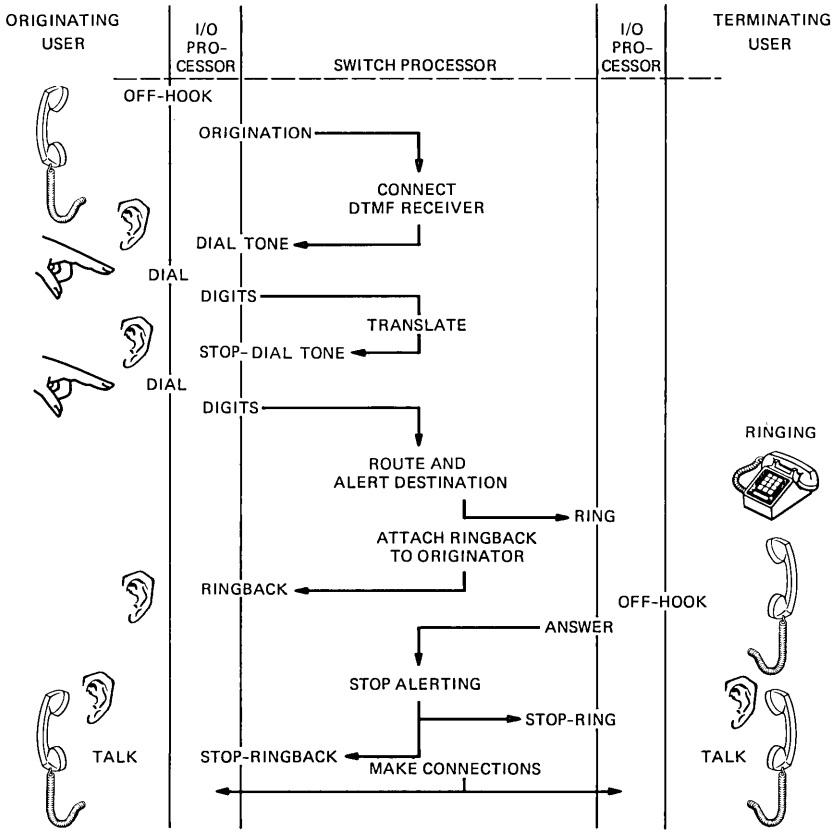


Fig. 1—Basic call example.

III. ARCHITECTURAL MODEL

To introduce the design of the switching software, we will step through a basic telephone call and develop a call model. Then the System 75 realization of this call model will be presented.

3.1 Basic call example

A user originates a call by going off-hook on his/her phone. This change is detected by an I/O peripheral and an off-hook signaling message is sent to the switch processor (see Fig. 1). On receiving this message, the resources for processing this call are allocated and messages are sent to the switch network for connecting a Dual-Tone Multi-Frequency (DTMF)* receiver and giving dial tone to the user.

*Acronyms and abbreviations used in the text are defined at the back of the *Journal.*

The call processing software interprets each digit dialed by the user and routes the call to the terminating station when all the digits have been dialed. The call is signaled to the terminating user with ringing, and the call progress is indicated to the originating user with ringback tone. When the terminating user answers the call, the switching network is instructed to remove the ringing signal and the ringback tone, and establish a talking path between the originator and the terminator. Finally, when either the originator or the terminator goes on-hook, the call processing software tears down the circuit connection and deallocates all the resources associated with the call.

This example illustrates the major functions of System 75 in processing a basic call:

- Terminal Handling
  The voice terminals supported by System 75 range from a single-line analog station to a simultaneous voice/data station with display, multiple call appearances, feature buttons, and data module. A variety of trunks are used to interconnect with other switching systems or a central office switch.
- Resource Management
  In addition to terminals, there are other resources in the system that need to be managed. These include the DTMF receivers, the time slots for circuit connections, tone generators, and the internal software records for call processing, messaging, measurements, and call detail recording.
- Call Sequencing Control
  Of the more than 150 features suported, many involve complicated sequencing logic to bring a call from one state to another. For example, the call coverage feature specifies the selection of new call destinations (coverage users) if no answer occurs in a specified time interval at either the principal destination or the current coverage destination. A conference call is another example where, in response to a user's conferencing request, the internal records and the circuit connections of the two initially distinct calls are merged to establish a common talking connection for all the parties.
- Routing and Termination Selection
  Routing and termination refer to the selection of a terminating endpoint or set of endpoints for a call. There are a wide variety of algorithms for routing and termination selection, such as hunting, bridging, coverage, least-cost routing, and routing data calls through pooled modem resources.

3.2 Call model

By analyzing the dynamics of the software functions in the above

list of functions, we can derive a call model that contains three major components: the *call*, the *group*, and the *user*. Highest in the hierarchy is the call, which ties all the parties of a connection together. Next is the group, which appears as a party on the call and contains a set of users. The user is an entity that models a terminal or a set of terminals that belong to a system user. This hierarchy is illustrated in Fig. 2. Let's examine the call, group, and user concepts in more detail.

3.2.1 The call

The call is associated with a set of connected parties. It is defined by a record of these parties plus the sequencing control logic of the call. It resolves asynchronous actions of various parties and directs the system's responses to these actions for the various feature operations. The call abstraction separates the common sequencing logic and information of a call from the group feature operations and the terminal handling.

3.2.2 The group

The group models a collection of users who are associated to provide special call and feature operations for the customer. There are several group types in System 75; each one contains special algorithms for the operations of that group type. For example, the hunt group specifies how a user should be selected from a group to receive a call. The group abstraction hides the internal operation of group features from the call, and provides a uniform structure for handling such group-oriented features as hunting, bridging, multiple attendants, and trunk groups.

3.2.3 The user

The user models the end user in the system, who may have a single telephone instrument or possibly a collection of interacting terminals. The user abstraction is a terminal handler that provides a set of

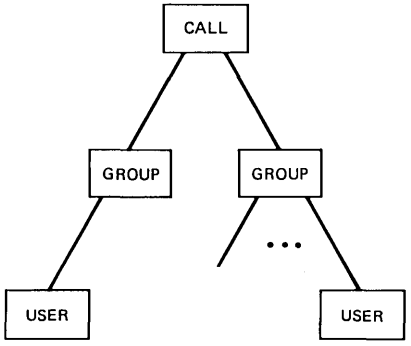


Fig. 2—Basic call model.

resources for which the different services compete: voice and data channels, status indicators, displays, ringers, etc. Trunks and data lines are also modeled as users. The intent of the user abstraction is to hide the terminal-specific operations from both the group and the call operations.

3.3 Realization of call model

The functional modules and dynamics of the call model are mapped into a software realization consisting of a layered set of cooperating processes. The processes execute under the Oryx/Pecos operating system[1] on the switch processor of the System 75 control complex.[2]

The call is realized by a call record and a single, transaction-oriented process that controls a call from origination to completion. A set of call processes exist to handle multiple calls in the system.

The group and user are realized by a group-manager and a user-manager process. Conceptually, there is a process per instance of each different group or user; in practice a single group-manager and a single user-manager process are multitasked to handle all instances of groups and users. The group-manager and user-manager processes provide a set of primitives that are independent of the group and user types of the system. These primitives serve as the building blocks for the upper-level software (e.g., the call process), where the feature sequencing is implemented.

This call model is a hybrid of both the process-per-call, or transactional structure,[3] and the process-per-terminal, or functional structure of switching systems.[4] The call process uses a process-per-call approach, whereas the group and user managers support the process-per-terminal approach. This hybrid design permits synchronizing the interactions of various terminals, as in the process-per-call approach, and permits isolating and distributing the terminal processing software, as in the process-per-terminal approach.

IV. SWITCH SERVICES SOFTWARE STRUCTURE

So far we have presented the call-process, and the group-manager and user-manager processes. There are additional processes that provide messaging and station services and the network and resource management functions of the switch. To organize the software, we define a service-control layer, a resource layer, and a driver layer of software (see Fig. 3). Within each layer, the various processes provide further information hiding and separation of functions.

4.1 Service control layer

The Service Control Layer contains a Service Dispatcher (SD) process and a process for each of the different services of System 75.

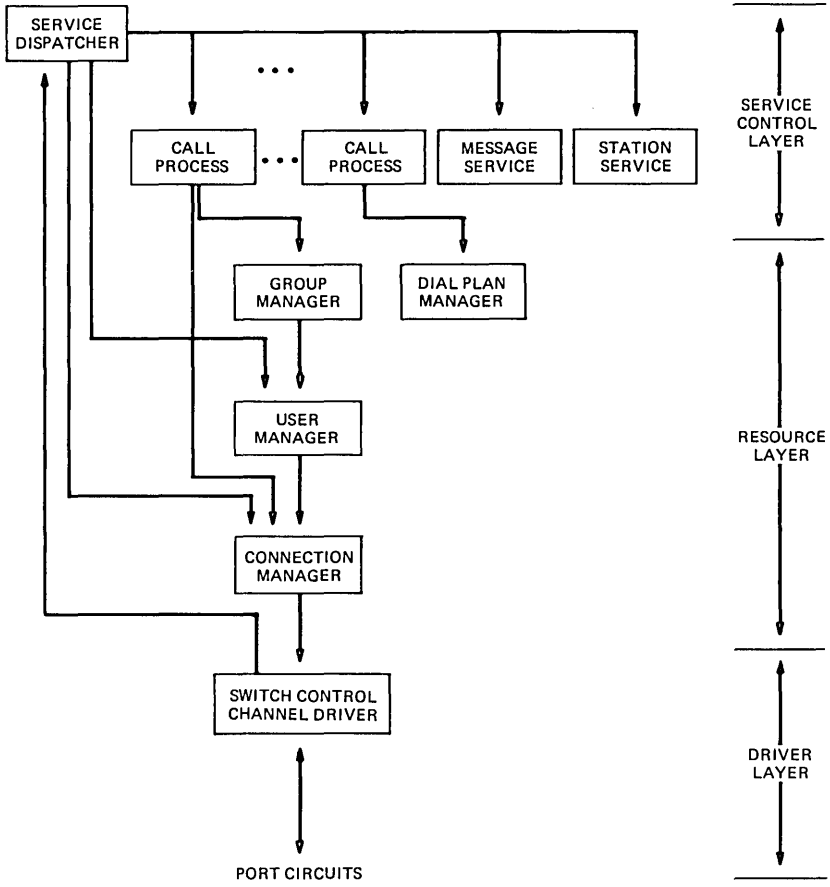


Fig. 3—Switch services software structure.

The service processes execute concurrently under control of the service dispatcher. User actions are translated into service commands by the resource layer and these commands are executed by a service process through primitives supplied by the resource-layer processes. Attention is given to the proper synchronization of user actions to guarantee that out-of-sync situations and deadly embraces cannot occur between concurrent transactions.

4.1.1 Call service

The Call Process (CP) provides the control and sequencing logic for call set-up and take-down and for a variety of feature operations in the system. Since a process-per-call is costly in terms of system resources, only a limited number of call processes exist to serve all calls. A call process is allocated to each call in a transient (e.g., dialing)

state and is deallocated from calls that have reached a stable (e.g., talking) state. The information is saved in a call record in the service dispatcher. The service dispatcher manages the pool of call processes, creating, allocating, suspending, and resuming them to/from calls.

There are many primitives supported by the resource-layer processes. The function of the call process is to invoke these primitives in the proper order and thereby produce the required response to the external user. It does this by analyzing the service-command message together with the current call state (e.g., idle or talking), and then invoking the appropriate sequencer code for that phase of the call. A call sequencer handles a special set of operations such as routing, answer, or drop (see Fig. 4). The operations are performed by invoking the primitives of the resource layer, which results in the driving of the hardware circuitry.

In addition to managing the call processes, the service-dispatcher process receives the signaling messages from the network and forwards them to the appropriate terminal handler process for interpretation into service commands. The appropriate service process, e.g., the call process, is then dispatched to execute the service command.

4.1.2 Message service

The control for the messaging services such as leave-word-calling and manual-message-waiting is provided by the Message-Service (MSG) process. The message service is a permanent server process, providing service for all users in the system. Message services can be invoked from a call by user terminal input or by other services. When message services are accessed via a call, the message-service process interacts with a call process. The message-service process accesses the

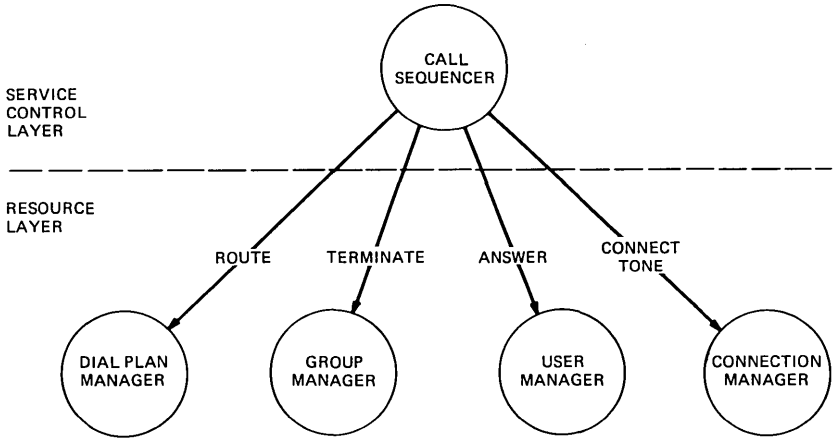


Fig. 4—Examples of resource layer primitives.

resource-layer processes for terminal input/output and translation data access.

4.1.3 Station service

The Station-Service (SSV) process provides miscellaneous station services such as integrated directory service, time-of-day display service, and programming some translation data from the user's terminal. The station-service process is a permanent server, handling all users in the system, similar to the message-service process. The station-service process accesses the resource-layer processes for terminal I/O and data access. When station services are accessed via a call, the station-service process also interacts with a call process.

4.2 Resource layer

The resource layer provides general resource management for the services, service-specific functions, and the line/terminal signaling. The system resources managed include the switch network, DTMF receivers, tones, trunks, telephone terminals, data terminals, groups, and databases like the system dial plan and the name/number directory. Call routing, queueing, terminal administration and maintenance, and feature activation primitives are some of the service-specific functions provided.

The software at the resource layer is functionally organized around the user and group concepts, the switch network, and the routing and directory database. A process exists for each of these main functions. Primitives are provided to the services by these processes for resource access. These primitives use the synchronous message facilities of the operating system to support the processing of user actions.

4.2.1 Group management

The Group-Manager (GM) process has all the translation data for group membership and group properties, and it maintains the state of the group and its members. Service-specific functions are provided by the group manager to manipulate the groups for the different services and features of the system. The group executes the service command on the users or members in the group according to the type of group. For example, in response to a call service terminate command, the coverage group would sequence the termination to each idle member in the group (an alert-all algorithm), while the uniform-call-distribution hunt group would select the longest idle member to receive the call.

4.2.2 User management

The User-Manager (UM) process contains both the user and terminal management software and status information. It presents an

abstract user or virtual terminal interface to the upper layers of software, while handling the signaling with terminals at the Driver Layer. Terminal access contention between the switch services, maintenance, and system administration is arbitrated by the user manager.

The user manager provides service-specific primitives to manipulate the users and terminals in the system. For example, the terminate primitive would, for a user with a multibutton telephone, select an idle call button, update the button-status lamp to flashing, and start ringing the telephone. The lamp and ringer operations are performed by sending signaling messages to the port circuit that interfaces the telephone.

Signaling-message interpretation is done by the terminal-handler functions in the user manager. Low-semantic-level messages, e.g., off-hook or button-push, are interpreted into service-level commands like originate and answer. A service-control-layer process then executes these commands by invoking the corresponding primitive provided by a resource-layer process.

4.2.3 Data management

The Dial-Plan-Manager (DPM) process provides access to and interpretation of translation databases in the system, including the system dial plan, the name/number directory, user permissions, least-cost routing patterns, and speed-calling numbers. Customized access to the data is provided for the call service. For example, digit analysis and the choosing of the initial routing destination are performed by the dial plan manager in response to a request from a call process.

4.2.4 Network management

The Connection-Manager (CM) process is responsible for the management of the network resources and for network control signaling. It abstracts the physical characteristics of the switch network and provides network connection primitives. Primitives to access the network resources like the DTMF generators and receivers are provided. Contention between the switch services, system maintenance, and system administration for the network resources is arbitrated here.

The communications network in System 75 is distributed onto each port board.[1] Network control messages to do time-slot assignment for listening and talking, gain adjustment, and six-party conferencing are sent by the connection manager to the port boards.

4.3 Driver layer

The driver layer encompasses the operating system drivers and the firmware in the intelligent port circuits of the System 75 communications network. The drivers include the Switch Control Channel

Driver (SCD), asynchronous data channel drivers, a timer driver, and bus drivers. The SCD interfaces the switch processor to the network control channel of the Time Division Multiplexed (TDM) bus (see Fig. 5). The network control (or archangel) acts like a full-duplex message switch between the switch processor and the microprocessors in the port boards (commonly referred to as angels). Error correction is provided between the SCD and the angels and flow control is managed by the archangel.

A functional message set is used between the switch processor and the angels. Signaling and network control messages are sent from the user and connection manger processes to the port circuits. The signaling messages are common across all types of ports. This helps to

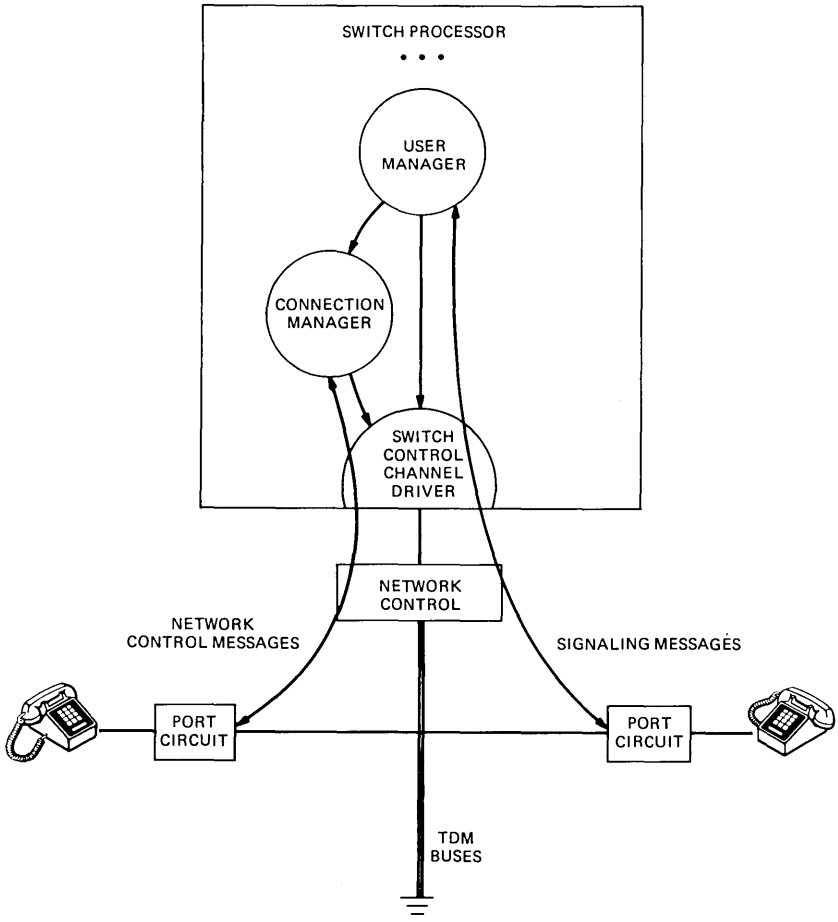


Fig. 5—Communications network control.

isolate the user manager from terminal-specific differences, providing the first level of terminal abstraction.

The angels off-load the switch processor by executing the low-level port-scanning, driving, and timing functions. This includes the ringer, tone, and lamp cadences, button scanning, and the timing for trunk signaling.

4.4 Call walk-through

A high-level walk-through of a call to aid in understanding the operation of the switch services software is presented here. The call is a station-to-station call between multifunction stations.

4.4.1 Origination

A station user goes off-hook with an idle call button selected, indicating the user wishes to "originate" a call. The off-hook signaling message is received by the service dispatcher, which forwards this signaling message to the user manager for interpretation. The user manager examines the terminal and user state and returns a call service originate command for the user back to the service dispatcher. The service dispatcher then allocates a call record, assigns an idle call process to this new call, and forwards the service command to the call process.

The call process enters the call setup mode, first requesting the connection manager to reserve network resources for the new call and then commanding the group manager and user manager to originate for the user. The user manager requests the connection manager to connect the originating user to the call and handles the lamp indications to the originating user. The origination message sequences are shown in Fig. 6.

Next, the call process requests the group and user managers to collect digits, or get a destination for the call. The user manager determines how to collect digits for this type of user and starts the collection, typically by requesting a DTMF receiver from the connection manager. Finally, the connection manager is requested to connect dial tone on the call. This finishes the setup part of the call and the user is now in a digit-collection state.

4.4.2 First digit

When the digit is received by the service dispatcher, it is forwarded to the connection manager, which has data about the call that the DTMF receiver is connected to. The connection manager returns a message that indicates a digit and the call number the digit is for. The service dispatcher forwards this message to the call process allocated to this call. The call process removes dial tone from the call and then

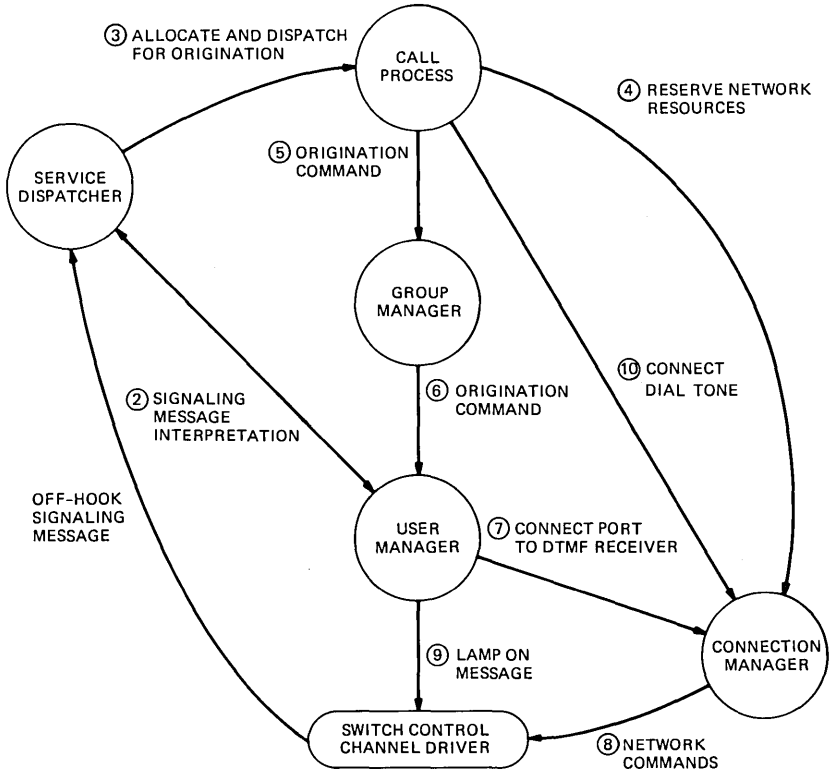


Fig. 6—Call origination sequence.

requests the dial plan manager to route the call based on the collected digit. The dial-plan manager returns data on how many more digits need to be collected. The call process then waits for the required number of digits or for an interdigit time-out or a hang-up.

4.4.3 Last digit

The last digit of a call is the signal that triggers the selecting and alerting of a destination. When the required number of digits have been received by the call process, it requests the dial plan manager to route the call based on the dialed digits. Permission checking is also done on the routing attempt. If enough digits have been dialed and the dialed destination is valid, the dial plan manager responds with the initial destination user. The call process then stops digit collection and the user manager releases any digit-collection resources from the call, such as the DTMF receiver.

Next, the call process attempts to terminate (i.e., bring in) the call to a destination user by requesting the group manager to terminate

the call. If the destination is simply a user, no further routing need be done, so the group manager requests the user manager to terminate the call. The user manager alerts the local user to the incoming call and replies to the call process that the user accepted the call. The call process then requests the connection manager to apply ringback tone to the originating user and "feeds back" the state of the termination and the identity of the destination to the group manager, which passes the information to the user manager—this results in the originating station receiving a display update if the user has a display.

The call is now in the terminating state, awaiting either an answer by a destination, an abandon by the originator, or a time-out to trigger the next stage of routing. Because the call is in a stable state, the call process is deallocated from the call.

4.4.4 Answer

The call is answered when a destination party goes off-hook. The off-hook is handled like the origination off-hook, except that the user manager determines that the user is now answering a ringing call. The service dispatcher allocates an idle call process and passes the answer command to it. The call process retrieves the call record from the service dispatcher and enters the call answer sequencer. First, any routing timers are canceled, ringback is removed from the connection, and the call process informs the originating user, via the group manager, that the call has been answered. Next, the call process commands the group manager to answer the call for the user. The group manager handles all users being alerted for the incoming call and commands the requesting user to answer the call. The user manager handles the answer command from the group manager by stopping the alerting indications at the user's station and commanding the connection manager to connect the answering port to the call.

Since the calling user was connected in the origination phase, the calling and called parties now have a two-way connection. The call is in a stable state so the call process is deallocated from the call.

4.4.5 Drop

The call is dropped when a user hangs up. The on-hook signal is processed in the same way as the off-hook, with the user manager interpreting the on-hook (or a button push) into a drop command. The call process receives the drop and commands the group manager to drop the user off the call. The gruop manager forwards this to the user manager, which then handles the lamp indications to the user and also commands the connection manager to remove the port from the connection.

Since there is only one party left on the call, the call process

sequences the teardown of the call by sending drop commands to the group manager. The drop commands are forwarded to the user manager, which again changes the user's indications and commands the connection manager to remove the port from the connection. Since all parties have been dropped, the call process informs the connection manager that the call no longer exists. The connection manager idles its connection and resource records for that call. The call process then cleans the call record and tells the service dispatcher to free it and the call record from the call.

V. ACKNOWLEDGMENTS

The System 75 switch services software architecture reflects the efforts of many people within the AT&T Information Systems Laboratories. Many ideas were the fruits of exploratory work that preceded the development phase.

REFERENCES

1. G. R. Sager, J. A. Melber, and K. T. Fong, "System 75: The Oryx/Pecos Operating System," AT&T. Tech. J., this issue.
2. L. A. Baxter et al., "System 75: Communications and Control Architecture," AT&T Tech. J., this issue.
3. Wing H. Huen, "What Is Different About Operating Systems for Telephone Systems," IEEE Reprint CH1515-6/79/0000-0179.
4. L. E. McMahon, "An Experimental Software Organization for a Laboratory Data Switch," Proc. ICC '81 (June 1981).

AUTHORS

Wayne Densmore, B.S. (Electrical and Computer Engineering), 1975, Clarkson College of Technology, M.S. (Electrical and Computer Engineering), 1976, Clarkson College of Technology; AT&T Bell Laboratories, 1976–1983; AT&T Information Systems Laboratories, 1983—. Mr. Densmore has been involved with software development for the *Horizon*® communication system prior to working on System 75. He is currently involved with enhancements to the System 75 architecture. Member, IEEE.

Ray J. Jakubek, B.S. (Electrical Engineering), 1967, Manhattan College; M.S., 1968 and Ph.D. (Electrical Engineering), 1973, New York University; Bell Laboratories, 1968–1982; AT&T Information Systems Laboratories, 1983—. Mr Jakubek has been involved in the design of a wide range of customer premises support systems and products. He contributed to the software development of the CNCC, NCOSS and RMATS II support systems. Subsequently he has supervised the software development of many of the call processing features of System 75. Since December 1982, he has been Head of the Software Applications department. Member, Eta Kappa Nu, Tau Beta Pi.

Michael J. Miracle, B.S. (Electrical and Computer Engineering), 1979, University of Wisconsin; M.S. (Electrical Engineering), 1980, Stanford University; AT&T Bell Laboratories, 1979–1982; AT&T Information Systems

Laboratories, 1983—. Mr Miracle worked on System 75 switch software development in the areas of call processing and data switching and is currently working on System 75 enhancements. Member, IEEE, Eta Kappa Nu, Phi Kappa Phi, and ACM.

John H. Sun, B.S. (Electrical Engineering), 1972, National Chiao-Tung University, Taiwan; M.S. (Computer Science), 1977, University of Connecticut; Taiwan Telecom Research Laboratories, 1974–1975; ACCO-Bristol Company, 1978; Bell-Northern Research, 1979; AT&T Bell Laboratories, 1980–1982; AT&T Information Systems Laboratories, 1983—. Mr. Sun has contributed to System 75 switch services software architecture, design and implementation both as a developer and in his current supervisory capacity.

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

System Management

By H. K. WOODLAND,* G. A. REISNER,* and A. S. MELAMED†

(Manuscript received July 11, 1984)

System management is the task of installing, administering, and maintaining a communications system. Customer and technician access to software-based administration and maintenance capabilities in the System 75 office communication system occurs through an on-line video display terminal called the System Access Terminal (SAT). With the SAT, the user may install, test, rearrange, and change equipment and services. The SAT hides the internal complexity of the system while presenting all the capabilities in as simple a manner as possible. A layered software architecture is used to perform data view mapping from the user view to the internal data representation.

I. INTRODUCTION

An integral part of System 75's enriched feature set is its capability of allowing the user to install, test, rearrange, and change equipment and services, and select a large number of per-user and system feature options. The user accomplishes this by entering and modifying data in the system's distributed, *translation* database. Also available are system-generated measurement reports, translation data backup on cartridge tape, and bulk data transfer (translation data and program updates) to AT&T Information Systems Operations Support Centers.

*AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc. †AT&T Bell Laboratories.

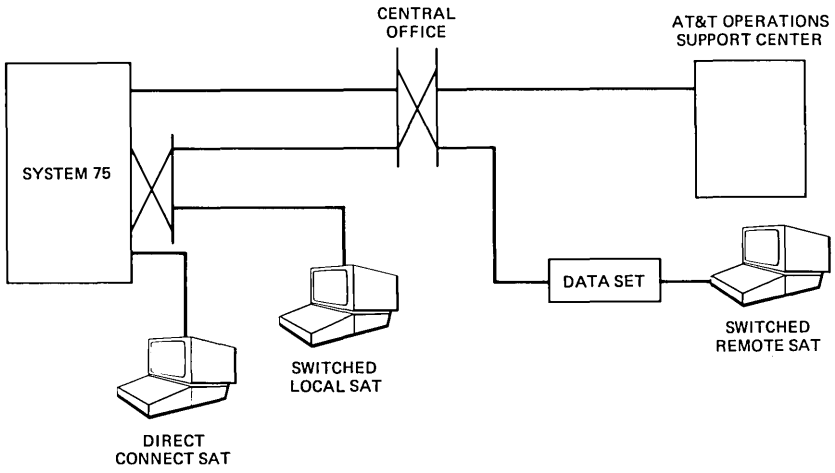


Fig. 1—System management access.

One objective of the system management task of System 75 is to allow customer participation by providing an interface that can be used by customer personnel as well as AT&T Information Systems technicians. This interface hides the internal complexity of the system while presenting all the capabilities in as simple a manner as possible.

This article discusses the key concepts of this user interface and the layered architecture that makes reliable translation database management possible.

II. TERMINAL USER INTERFACE

2.1 System access

System 75 operations and training are simplified by providing a single system-management user interface. Customer and local technician access occurs through an on-line video display terminal called the System Access Terminal (SAT).* The SAT (the 513 or 515 BCT) may be:

1. Directly connected to an RS-232C EIA port,
2. Switched through a Digital Communications Protocol (DCP) dial-up port, or
3. Switched through a modem dial-up port for off-premises access (Fig. 1).[1]

The local SATs operate at 9600 baud while the remote SAT—connected via a Direct Distance Dialing (DDD) line—operates at 1200 baud. In addition to the SAT interface, a 1200-baud, X.25, host-to-

*Acronyms and abbreviations used in the text are defined at the back of the *Journal.*

host interface is provided to accommodate the remote AT&T Information Systems Operations Support Center.

2.2 Terminal user access

Access security, an important aspect of any shared access system, is maintained through the use of login names and passwords. Repeated invalid passwords will cause the system to drop the line and increment a security violation count.

Multiuser access is also important. System 75 allows two SAT users and one remote operations host to be logged in at the same time. User contention is managed on a per-command basis and only one user at a time may execute a command that changes the database. This eliminates the confusion that exists when two users interact with the same data. These design requirements greatly simplified the architecture and implementation.

2.3 Terminal interface screen layout

A simple, consistent, easy-to-understand screen layout is an essential element of a good user interface. The System 75 terminal interface screen is divided into four regions (Fig. 2):

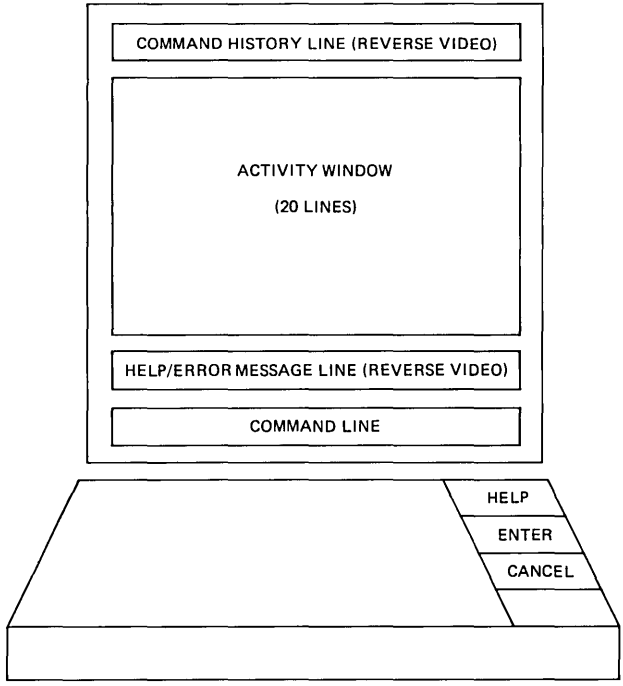


Fig. 2—Terminal interface screen regions.

1. A command path or *command history line*
2. A 20-line command output or *activity window*
3. A *help/error message line*
4. A *command line.*

Reverse video is used in alternate regions to partition the screen layout without wasting valuable screen lines.

2.4 Command entry

Command organization is another important aspect of a good user interface. Upon logging in, the System 75 user is placed in a single-level *command entry state.* All commands are directly accessible from this state. There is no tree structure to traverse or mode keys to depress to move among administration and maintenance commands.

Commands are logically divided into ten "administrable" categories. Each user login name is allowed or denied permission to execute commands on a per-command category basis. Confusion is eliminated because a user only sees those commands that he has permission to execute. As the need and desire for customer participation increases, the customer may be given permission to execute more commands.

The commands utilize English-like phrases. The format is *action, object, qualifier.* For example, the command to add a station set with the extension number 3261 is:

```
add station 3261,
```

where `add` is the *action* to be performed, `station` is the *object* being acted on, and `3261` is the *qualifier.* Complete or partial commands may be entered providing the user enters enough characters of a parameter to distinguish it from other legal parameters. For the example above, the command could be entered as:

```
a s 3261.
```

The system prompts the user with on-line messages and, at any time, the user may request assistance by depressing the *help* key. The response from the system depends on the parameter type to be entered: if an *action* or *object,* then a list of valid actions or objects is displayed; if a *qualifier,* a message is displayed describing the format and type of data to be entered. For example, if the command was

```
add station 20,
```

the system would respond with the message

```
20 is an invalid entry—Please Press Help.
```

If the *help* key were depressed, the system would respond with the message

`Enter a number between 100–9000,`

where `100–9000` are valid extension numbers. This sort of prompting greatly reduces training time and paper documentation. Other actions include `display`, `change`, and `remove`.

2.5 Command output

Three basic types of output appear in the *activity window*:

1. A status report,
2. A data entry form, and
3. A restricted form.

A *status report* is produced when a `display` command is entered, a data entry form is produced when an `add` or `change` command is entered, and a restricted form is produced when a `remove` command is entered. In the *data entry form state*, the user may modify the data and then store away the changes by depressing the *enter* key. In the *restricted form state*, the user is prohibited from modifying the data on the screen. The data are displayed only to allow the user to verify that the correct data are being removed. The user may depress the cancel key to abort any operation.

2.5.1 Data entry form state

The *data entry form state* allows formatted, forms-based data entry. Forms are associated with a specific task. For example, the station form is used to display the precise data required to add a station set (Fig. 3). Fields are added or removed from the form dynamically, as required. When the data entry form state is invoked, a form appears in the activity window with data fields defaulted to the most commonly

```
add station 3261                                          Page 1 of 1

                                    STATION

 Extension:  2000            Station Type: 7303S              Port: _____
      Name: _____________      Coverage Path: __                 COS: 1_
                               Security Code: ___                COR: 1_
FEATURE OPTIONS
            LWC Reception? y        Coverage Msg Retrieval Permission? y
          LWC Activation? y                          Data Restriction? n
     Redirect Notification? y                 Idle Appearance Preference? n


ABBREVIATED DIALING
     List1: _______              List2: _______              List3: _______

BUTTON ASSIGNMENTS
  1: call-appr                          6: ________
  2: call-appr                          7: ________
  3: call-appr                          8: ________
  4: ________                           9: ________
  5: ________                          10: ________
```

Fig. 3—System management station form.

used values. The command entered and the number of form pages appear on the *command history line.*

Standard cursor control keys (arrow, next/previous field, clear field, next/previous page, and refresh screen) allow the user to move about the form in a prescribed way. In the station form, the *next field* key moves the cursor left to right in the upper part of the form, and down the columns in the bottom (button assignment) part of the form. This guides the user to complete the form in an orderly manner. However, the *arrow* keys provide the flexibility to move the cursor in any desired direction.

2.5.2 Data validation

In some systems, data are validated only after all data have been entered. Where possible, System 75 performs validation immediately as data are entered or changed (see Section 3.4). This allows immediate response to many user errors, thereby decreasing the overall user time required to enter and validate a given form.

2.6 Special commands

Two special commands were added to assist the user in adding objects to the system. One handles the case when the user wants to add an object and does not know the next available qualifier. For this case, the `add object next` command is provided. For example, if `add station next` is entered, the system searches the database, selects the next available extension number, and invokes the data entry form state with the selected extension number.

The *duplicate station* command speeds the addition of a station by copying data from an existing station. For example, if `duplicate station 3261` is entered, the system invokes the data entry form state and displays a station form with data identical to those of station 3261, except that the extension, port, and name fields are blank. These differentiating fields may then be entered to create the new station.

In summary, the SAT provides a simple, easy-to-understand user interface that may be used by the AT&T Information Systems technician and trained customer personnel.

III. SYSTEM MANAGEMENT ARCHITECTURE AND DESIGN

The system management software provides four functions, all of which are available through the SAT:

1. Measurement collection and reporting
2. Maintenance testing and reporting
3. Translation data backup on tape
4. Translation database management.

The measurement collection and reporting capability provides

hourly traffic data on engineered resources, e.g., trunk groups, which are made available through formatted reports. Maintenance reporting and testing capabilities include the demand testing of circuit packs and terminal equipment and the display of system error and alarm logs. Translation data backup on tape provides system translation data backup on a secondary storage medium. Underlying all system management functions is translation database management. The remainder of this article concentrates on this topic, and the software designed to support it.

Translation database management software provides four important functions: data view mapping, database validation, form transactions and concurrency control. Data view mapping allows a user to display and change all translation data related to a single task (e.g., station installation), while hiding the complexities of the internal data organization. Database validation ensures that data entered into the system are individually correct and consistent with respect to other data. For example, validation ensures that extensions assigned to stations are consistent with the dialing plan. Transactions ensure that all the translation data entered on a system form are either valid and accepted or inconsistent and rejected. Concurrency control allows there to be multiple SAT users and, at another level, ensures that switch services software will not use critical data that are being changed. Before describing how these functions are accomplished, we provide a brief overview of the system management software structure. The software consists of three layers:

1. User interface and control
2. Command execution and validation
3. Data access and storage.

The *user interface and control* layer provides users with access to the system, through either the SAT or the X.25 remote link. The *command execution and validation* layer and the *data access and storage* layer provide a single, record-based interface to the user interface and control layer, independently of the access method. Figure 4 shows a block diagram of the essential software layers and components.

The *command execution and validation* layer consists of four software modules, each of which supports the different logical functions described above: (1) measurement collection and storage, (2) administration database update/validation, (3) maintenance command execution, and (4) translation backup on tape.

The *data access and storage* layer consists of the administration database access module and all processes which store translation data.

3.1 Data view mapping

Translation data in System 75 are distributed. They are contained

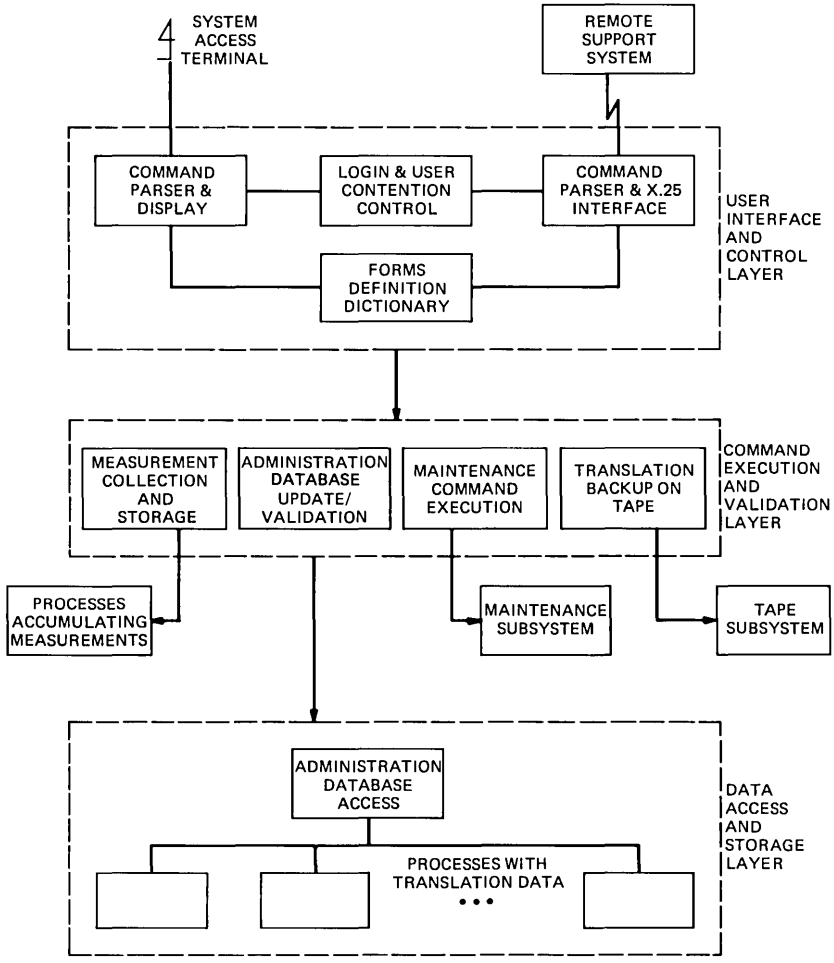


Fig. 4—System management layered software.

in the local data space of the processes that access them in order to meet real-time requirements for voice and data switching. This results in the fragmentation of logically related data. System management software maps the data entered on forms (the logical view of translation data) into the internal view of data, as required by the switch services architecture.[2]

3.1.1 Low-level view of data

The administration database access module distributes and retrieves translation data using a record-based interface. This is accomplished by a set of C language library functions that are supported by each

process storing translation data. This interface is akin to the data manipulation language of database management systems. At an abstract level, however, this interface is designed to be less powerful than a true database management system (e.g., relational manipulations on internal tables are not supported). The reason for this choice of design is to reduce complexity and generality of access methods, thereby increasing the speed of data retrieval.

A simplified view of low-level data view mapping is shown in Fig. 5. It consists of two parts: mapping external identifiers to internal identifiers and distributing data values to the processes storing translation data. This concept is illustrated by looking at some of the data associated with a station set (see Fig. 6). The external identifier for a station is its extension, i.e., the number dialed to call the station. Some of the data associated with this station are the type (digital), name of the person assigned to the station (John Doe), and various buttons.

The internal identifier for the station is an ordered pair (TYPE, INDEX), where TYPE=STATION identifies the station table. This internal identifier is generated by finding the first empty slot in the station table. The internal identifier is then used as a key to identify other data for the station in the button table and the miscellaneous data table. The mapping from extension to internal identifier is contained in the extension table (this extension-to-internal identifier table is used whenever a station extension is dialed).

In summary, the mapping between internal and external identifiers is accomplished completely within the data access and storage layer.

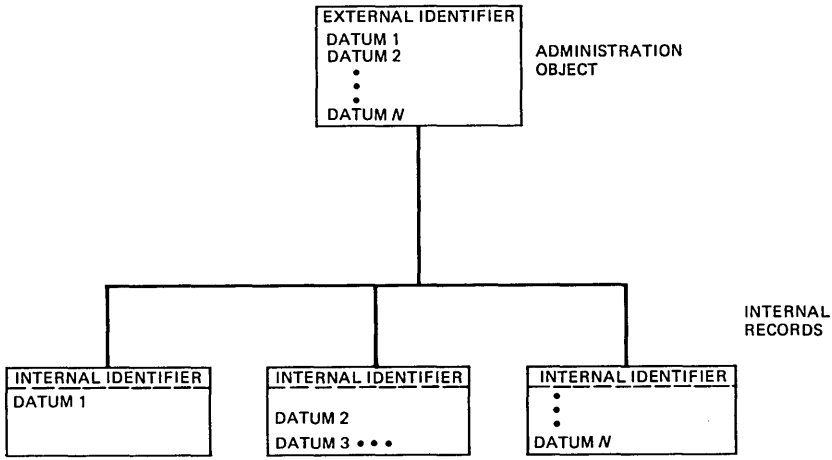


Fig. 5—Simplified low-level data decomposition.

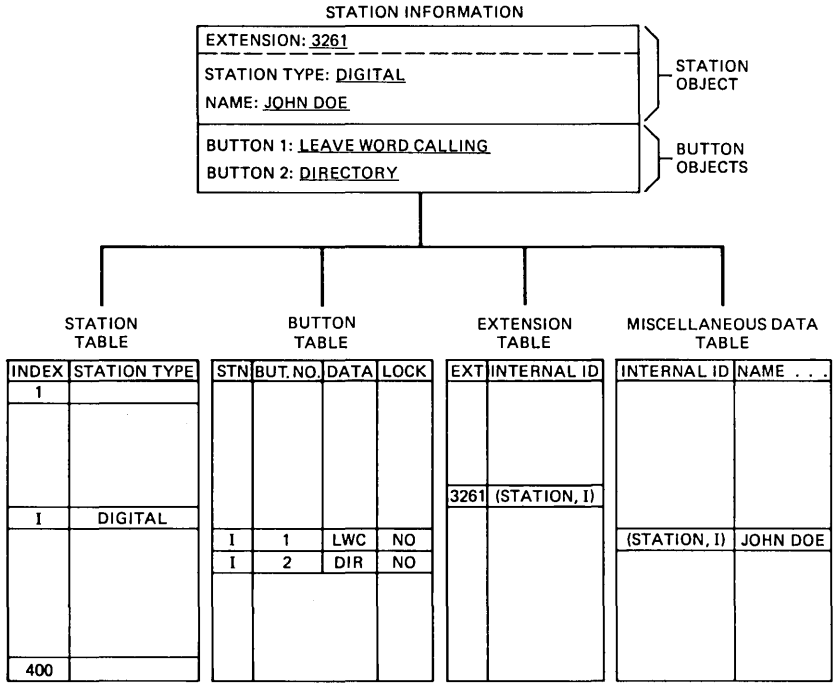


Fig. 6—Data decomposition for a station.

The upper layers of system management software use external identifiers exclusively.

3.2 Concurrency control

As in any database system with multiple users, System 75 must provide a means of managing concurrent updating and a means of managing simultaneous reading and writing of translation data. The first concern, concurrent updating, is handled at a very high level in the user interface and control layer. The login and user contention control module tracks each user logged into the system and each command executed. In this way, it assures that only one user at a time is trying to update translation data. It performs this control by allowing only one `add`, `change`, or `remove` command to be executed at a time. Thus, the problem of multiple concurrent database updates is eliminated.

3.2.1 Locking of internal records

The second problem, managing simultaneous reading and writing of data, is handled in the data access and storage layer. To solve this problem System 75 implements read/write locks on internal records.

In any system using locks, there is a trade-off between speed and the overhead of the locking mechanism. Because System 75 is a real-time switching system, and because data retrieval involves the very same processes that are accomplishing switch services, locks are used with a small, selected subset of internal records, thereby eliminating unnecessary performance penalties.

The mechanism for locking is illustrated in Fig. 6, the station data decomposition example. A read/write lock is associated with each button in the database. This lock serves two purposes. First, it prevents the switch services software from using button data while the button is being changed. This assures that the status data maintained for each button/lamp is consistent with the translation state. The consequences of inconsistencies could manifest themselves, for example, in a lamp that is stuck in the lit state permanently (for a button that no longer has a function associated with it), or until the station is reset by unplugging and reinstalling or until fixed by an internal system audit. The second purpose allows changing an individual button on an active station as long as that particular button is not in use. This granularity of locking provides a great deal of user flexibility in changing data while also guaranteeing a high degree of data protection. Locking also supports a database transaction mechanism described in the next section.

3.3 Database transactions

Administration of the application database takes place during normal system operation. System 75 uses a database transaction mechanism to guarantee that the user command associated with a form is either processed to completion or not at all. In database terminology, transaction management maintains the database consistency, permanency, and atomicity.[4] The capability to abort transactions exists. When one of the requests of a transaction cannot be performed, an automatic rollback of the partially completed transaction occurs. The same procedure is followed in case of system recovery, provided a database transaction was in progress.

The system management software places *exclusive* locks and the switch services software places *shared* locks on data. The first request in a transaction encountering a locked resource causes the system to back out of the transaction, thus avoiding deadlock. A two-phase locking protocol is used. All the internal records required by a transaction are locked before any locks are released. The locks are released only after the entire transaction has been processed.

Transaction requests are generated by the command execution and validation layer, and the actual transaction mechanism is implemented

in the administration database access module. A typical transaction consists of multiple requests. For example:

```
BEGIN_TRANSACTION
UPDATE STATION (3261)
UPDATE STATION_BUTTON (3261, BUTTON 1)
UPDATE STATION_BUTTON (3261, BUTTON 2)
END_TRANSACTION
```

A single request from the command execution and validation layer may map into multiple operations on internal records. When a single request is received, the administration database access module dynamically allocates an instance of an internal record execution table, which depends on both the *object* of the request (e.g., station) and the *action* of the request (e.g., update). Figure 7 illustrates the transaction execution table for the transaction above. The execution table consists of a dynamic and a static part. The static part describes the sequence of

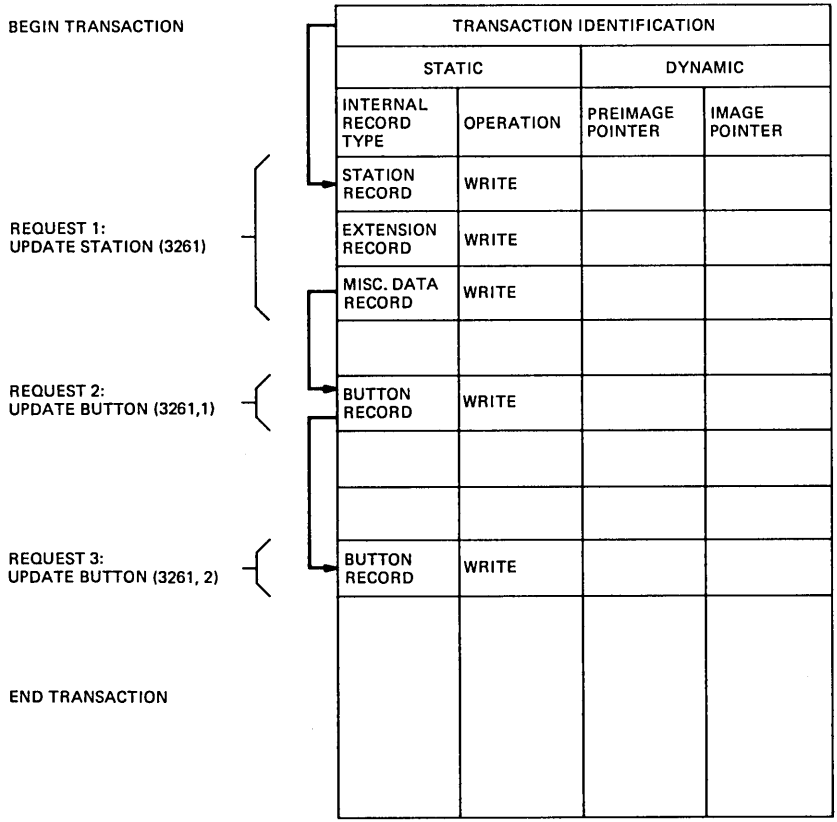


Fig. 7—Transaction execution table.

operations for this request (*object, action*) pair. The dynamic portion contains pointers to the preimages and images of the data to be changed. The preimages are used for possible recovery actions, i.e., for backing out of a transaction.

The instances of the *execution tables* (one per *command execution and validation* layer request) are stacked on behalf of a transaction. For each transaction in progress, the current pointer points to the last entry processed. When a transaction terminates successfully, all the tables are deallocated. The tables are used to perform transaction back-out and recovery.

3.4 Database validation

As we have seen, the *data access and storage* layer distributes system data to internal records, and this function is guaranteed to be robust because form transactions and a locking mechanism are utilized. To help simplify the software in the administration database access module, these functions are accomplished in a data-independent fashion wherever possible. The function of ensuring the correctness (i.e., semantics) of the database is accomplished in the administration database update/validation module.

Validation is accomplished as early as possible in the data entry process. Each field on a form is checked before the user is allowed to advance to the next field. This checking can be as simple as a range validation or as complex as checking that a station is installed before it may be entered into a hunt group, thus ensuring the correct operation of calls terminating to the hunt group. After all the necessary or desired information is typed on a form, the entire form is validated when the user depresses the *enter* key. At this time all interfield relationships are examined and verified. An example of this is ensuring that a multibutton station has at least two call-appearance buttons installed on it. Two buttons are necessary for the correct operation of the conference and transfer features.

IV. TABLE-DRIVEN SOFTWARE STRUCTURE

System management software in System 75 is a good example of the use of table-driven software. This table-driven approach is chosen for gains in memory usage and for ease of adding new administration objects. At each layer of system management software there are examples of modules which use this approach, such as the forms definition dictionary, administration database update/validation, and administration database access. To illustrate the approach, the software structure of the administration database access module, along with the tables that are used by this generic software, are briefly described below.

Generic software belongs to the following categories:

1. Table allocation
2. Request interpretation
3. Field conversion
4. Invocation of internal record library functions
5. Transaction control.

These five categories are executed, in the order shown, for any request from the command execution and validation layer. The precise software used is, for some categories, dependent upon the *action* (e.g., update or *add*) requested. However, this software is almost completely independent of the objects involved. Knowledge of object manipulation, conversion, and storage is contained in the following four major types of tables:

1. Request driver table—Contains all the actions that are valid for a particular object, along with pointers to other tables that are used for each request (*object, action*) pair.

2. Operation table—Provides a sequence of run-time operations to be performed on internal records for a given (*object, action*) pair. The order of execution is implicit in the order of operations in this table.

3. Field map table—Describes the field mapping between the individual fields of an object and the individual fields in associated internal records.

4. Internal record location table—Indicates which process contains each internal record.

An example of the use of these tables is shown for (station, update) in Fig. 8. Note that in the operation table the station record is listed first. This is critical for (station, add) since the internal identifier for the station is generated by the insertion of the station record. This internal identifier is needed in the other records associated with the station.

For some objects, the object-to-internal record mapping is data dependent, that is, the set of internal records that a particular object maps to depends on a particular transition of field values. These data-dependent transformations are handled using a *pseudo-object* concept. The basic idea is that *pseudo-objects* are used to enumerate all the different possible object-to-internal record mappings for a particular object. This enumeration enables the administration database access module to use the regular tables even for the data-dependent transformations. The *pseudo-object* concept is completely hidden from the higher-level software.

V. CONCLUSION

This article has presented two aspects of System 75 system management: the system access terminal and the layered software archi-

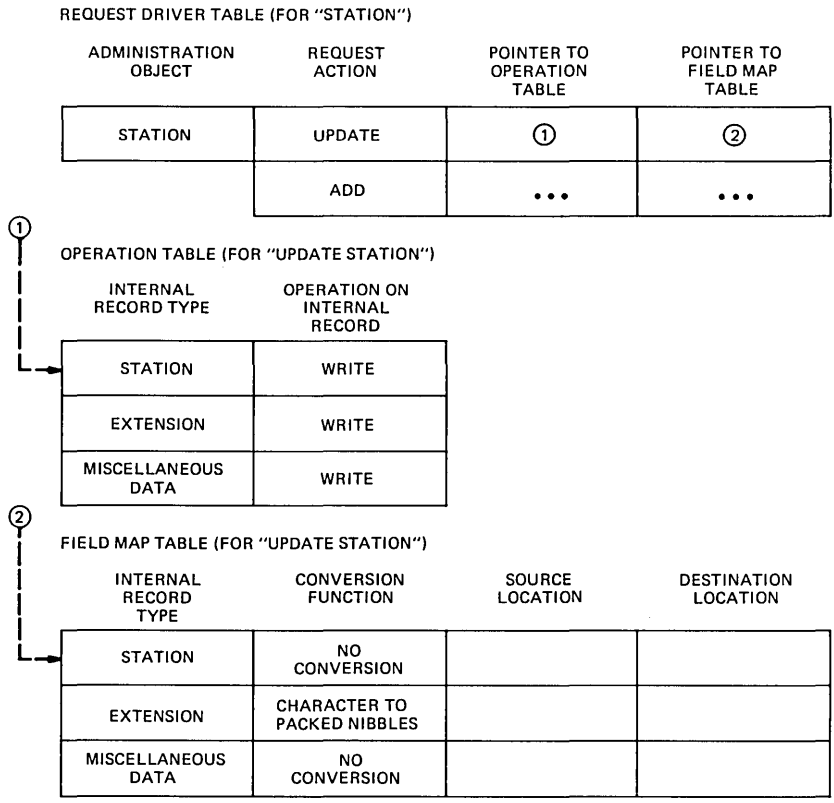


Fig. 8—Example of table-driven software.

tecture. The system access terminal provides a vehicle for the simple presentation of a complex database. The layered software architecture makes this all possible by providing data view mapping, database validation, transactions, and concurrency control. Moreover, it provides a structure that allows expansion to new functionality.

REFERENCES

1. L. A. Baxter et al., "System 75: Communications and Control Architecture," AT&T Tech. J., this issue.
2. W. Densmore et al., "System 75: Switch Services Software," AT&T Tech. J., this issue.
3. C. J. Date, *An Introduction to Data Base Systems*, 3rd edition, Reading, MA: Addison-Wesley, 1981.

AUTHORS

Anna S. Melamed, B.S.E. (Electronical Engineering), 1969, Warsaw Polytechnic; M.S.E. (Computer Information and Control Engineering), 1972, The

University of Michigan; M.S. (Mathematics), 1974, The University of Michigan; Ph.D. (Computer Information and Control Engineering), 1977, The University of Michigan; Bell Laboratories, 1978–1982; AT&T Information Systems Laboratories, 1983–1984; AT&T Bell Laboratories, 1984—. Ms. Melamed has been involved in the definition, design and implementation of various software systems such as a Test Language Parser, a database restructuring system for the AT&T 3B20-DMERT (Duplex Multiple Environment Real-Time) applications and system management for System 75. She is currently involved in performance analysis of a distributed computer system.

Gerald A. Reisner, B.S. (Applied Mathematics), 1967, New York University, School of Engineering; M.S. (Applied Mathematics), 1970, Adelphi University; Ph.D. (Mathematics), 1974, University of Minnesota; Hazeltine Corporation, 1968 to 1970; Bell Laboratories, 1974 to 1983; AT&T Information Systems Laboratories, 1983—. At Bell Laboratories, Mr. Reisner's initial assignment involved characterizing network blocking under various conditions, including retrials. He also provided performance measurement requirements for local switching systems. In 1978 he moved to software development for the Network Control Operations Support System, a minicomputer-based system used to install and maintain private networks. Since 1980 he has designed Switch Services Software for System 75, and since 1983 has been a Supervisor for System 75 Switch Services Software. Member, Tau Beta Pi, ACM and MAA.

Harold K. Woodland, B.S. (Physics), 1971, Morgan State University; M.S. (Engineering), 1973, Cornell University; Bell Laboratories, 1973–1983; AT&T Information Systems Laboratories, 1983—. At Bell Laboratories, Mr. Woodland's work has included design of special trunk circuits and software development for the *Horizon*® communication system Customer Access Unit, the Enhanced 911 Emergency Reporting System (E911), the *Horizon* ACD and System 75 System Management. Since 1981, he has been Supervisor of the System 75 System Management and Administration group.

AT&T Technical Journal
Vol 64, No. 1, January 1985
Printed in U.S.A.

System 75:

Maintenance Architecture

By K. S. LU,* J. D. PRICE,† and T. L. SMITH*

(Manuscript received July 11, 1984)

Reliable service has been a cornerstone of customer premises communications systems for years. System 75 office communications system hardware and software have been designed to continue that high degree of reliability and availability. The hardware has been designed to detect and correct errors as they occur, to minimize the number of components that cause system outage, and to simplify fault isolation to a replaceable component. The software has been designed to recover from intermittent failures and to continue providing service with a minimum of disruption. These features have been implemented in the software as a group of processes running under a real-time operating system, which simplifies building and testing the software and makes it easy to extend its functions.

I. INTRODUCTION

Reliable service is one of the most important features of a customer communication system. Aspects include reliable hardware design, automatic system reconfiguration in the event of a hardware fault, defensive programming to minimize the impact of intermittent hardware failures or obscure program bugs, and a repair strategy that quickly corrects any hardware problems in the system. System 75 is

* AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc. † AT&T Consumer Products.

based on previously proved designs[1-3] to ensure reliable service. It includes some new features as well. Major enhancements include a CRT-terminal-based, human-engineered interface to simplify repair activities; a maintenance system that provides systematic organization and flexibility to system maintenance activities; and features to improve the reliability of the communications network by maintaining connected equipment such as stations, terminals, and trunks.

This paper describes various aspects of system maintenance including objectives, the plans for hardware and software maintenance, the maintenance software architecture, a discussion of how the system handles critical failures, the human interface to the maintenance features, and the remote maintenance capabilities.

II. DESIGN OBJECTIVES

A clear set of objectives was required to build a consistent and integrated set of maintenance features in a timely way. Some of these were external objectives, observable by people who use or repair the system:

1. The system must be highly reliable. Any component failure should affect the smallest possible piece of the system, and the time between system outages caused by a single component failure should be several years. Automatic recovery from hardware and software problems should be achieved as well.

2. The system, not the users, should detect failures. The system should be fully self-testing with tests run frequently enough to detect failures before the user. False alarms should be avoided; alarm only those faults requiring repair.

3. The system must be simple and quick to repair. Trouble analysis, testing, and direction of the repair activity should be provided by the system, not the repair person. The system should have an interface that can be used with little training, permitting the customer to participate in maintenance.

4. The scope of the maintenance should extend to the entire customer communication system. The system should help maintain stations, terminals, trunks, and other connected equipment.

In addition to the external objectives, there are several objectives which, though not directly visible, impact maintenance performance:

1. The system should have a common set of tests for each piece of the system. By using the same test for system fault diagnosis, periodic testing, and technician testing, test results are reproducible and the possibility for confusion is reduced.

2. The maintenance system should have the flexibility to deal with a wide diversity of objects. By using a maintenance operating system

approach, maintenance of each piece of the system can be written independently while using common structure and functions.

3. The system should be easy to extend. As the features and functions of the system increase, the maintenance must also expand with minimal revisions to existing features.

4. The system should have minimal maintenance interactions between parts of the system. This makes it easier to identify the part of the system that is failing and permits inherently simpler maintenance strategies to be developed.

III. MAINTENANCE OBJECTS

Maintenance programs for System 75 run under the Oryx/Pecos operating system.[4] (See Fig. 1.) The collection of processes that perform the maintenance activities is called the maintenance subsystem and is written as a maintenance operating system. Each part of the system to be maintained is called a maintenance object and is handled independently by the maintenance operating system. Maintenance objects are selected to be independent from each other, reducing the number of interactions inside the maintenance system and simplifying the maintenance strategies. Each object in the system has its own maintenance strategy. This strategy can include tests for detecting

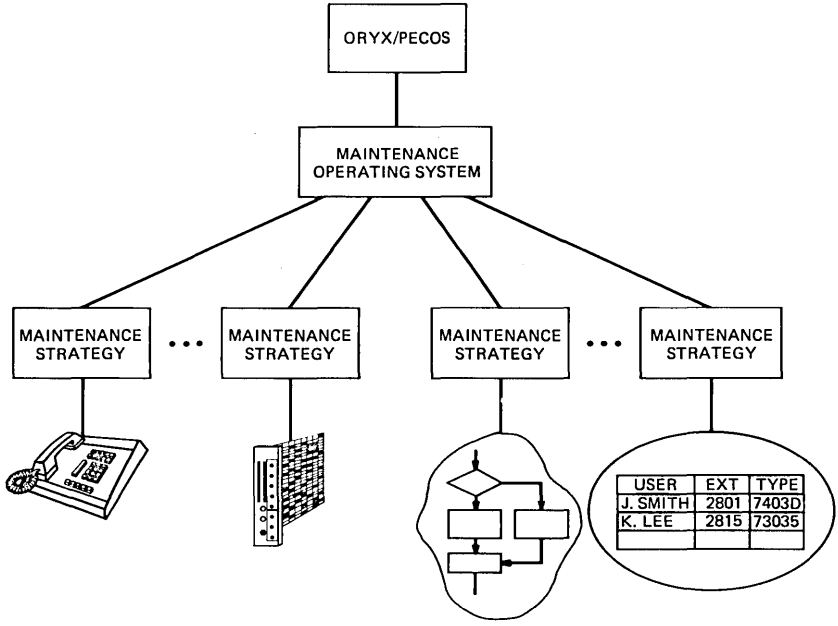


Fig. 1—Maintenance object organization.

and diagnosing problems, and recovery, reconfiguration, and repair activities. The maintenance operating system provides common functions such as scheduling, alarming and terminal interfaces.

There are three general categories of maintenance objects. The first category is hardware, including circuit packs, stations, and trunks. In general, each circuit pack or each separately replaceable unit is a separate maintenance object, but there are exceptions where several circuit packs belong to one maintenance object or where one circuit pack contains several maintenance objects. Hardware maintenance objects can be tested, alarmed, and removed from service. When a problem has been isolated to a hardware maintenance object, the object is replaced.

The second category of maintenance objects is software processes. Each process running under the Oryx/Pecos operating system is a separate maintenance object. If a process encounters trouble, it can be recovered or restarted.

The third category of maintenance objects is data relationships. Each group of data that has some internal structure or redundancy can be maintained. Data relationships can be audited and corrected.

IV. HARDWARE MAINTENANCE

4.1 Principles

4.1.1 Scope

Hardware maintenance covers all the equipment inside the cabinet including all the circuit packs (the processor complex, trunks, lines, and service circuits), the tape drive, the power converters, and cabinet-level functions such as power supplies, batteries, and environment (see Section 4.4.4). In addition the system maintains attached lines and trunks.

4.1.2 Error detection

Whenever possible, the system will detect and report errors automatically. Error detection is done through a variety of methods, with the particular method determined by weighing the severity of the problem against the cost of the method. The following methods of error detection are used:

1. Error detection hardware is added to the system, monitoring the operation and detecting errors immediately. This is cost-effective in only the most critical applications. An example is the error detection and correction circuitry on the processor memory. When a faulty memory location is accessed, the failure is detected immediately.

2. In-line errors are detected using software or firmware. Each time a circuit performs its function, additional software checks are run to ensure that the circuit is functioning correctly. This is useful in

important applications where quick detection of errors is worth the cost of processor real-time consumption. The checksum on control channel messages is an example.

3. Periodic tests are the most widespread testing strategy. Tests that run quickly or detect important troubles are run frequently, typically once an hour; a comprehensive set of tests is run once every 24 hours.

4. A few error-detection tests are potentially so service disruptive that they are not run on a periodic basis. A comprehensive memory test falls into this category. As long as memory is working correctly, no comprehensive test is run. However, if the system has experienced several crashes, additional time will be taken on the reboot to run a complete memory test.

4.1.3 System alarms

If a maintenance object consistently fails the error-detection tests, the system will automatically generate an alarm. This is a call for technician action to repair the system and restore it to a normal condition. There are three levels of alarms.

1. Warning alarms for failures that cause no noticeable degradation in the customer service. Also included are failures whose cause might be external to System 75 but need no immediate action.

2. Minor alarms for failures that cause marginal degradation of customer service while not rendering a crucial portion of the system inoperable. This condition requires action, but its consequences are not of a global or immediate nature. Problems might be impairing service to a few trunks or stations or causing problems to only one feature across the entire system.

3. Major alarms for failures that cause critical degradation of customer service and require immediate action. Processor faults and failure of an entire trunk group fall in this category.

Alarms are made visible in several different ways.

1. If the failing component is a circuit pack, a red lamp is lit on that circuit pack. This guides the technician in replacing the faulty circuit pack.

2. If the system detects any major or minor alarm conditions, they will be reported to the Operations Support Center computer.[5]* The system will automatically place a call and report the problem (see Section IX).

3. The system alarm status is displayed in lamps on the maintenance circuit pack alarm panel. A second lamp indicates the status of

* A typical centralized Operations Support Center computer system is RMATS (Remote Maintenance Administration and Traffic System).

attempts to notify the Operations Support Center computer of the problem.

4. If the system has any major or minor alarms, an alarm lamp on the attendant consoles will be lit.

5. A list of all alarms currently in the system is kept in an internal alarm log. This can be displayed on command from a system administration terminal (see Section VIII) or can be requested through the remote computer interface.

In addition to automatically raising alarms, the system will also automatically retire alarms. Sometimes a problem that is causing an alarm will disappear without human intervention. Since testing continues on an alarmed maintenance object, tests will begin to pass after the problem disappears, causing the alarm to be retired. This strategy eliminates needless repair activity to retire alarms for temporary faults. In addition, no technician activity is required to retire an alarm after repair.

The maintenance strategy is to eliminate intermittent alarms and to alarm only consistent, reproducible faults. This is done by requiring an error to be seen several times before it generates an alarm. Similarly, once a maintenance object is alarmed, it must pass its tests several times before the alarm is retired.

To generate alarms in a reasonably short interval of time, the system uses intensive testing. Once an error is detected on a maintenance object, that object goes into a testing mode where tests are run much more frequently. Either these tests will fail several times in a row and quickly generate an alarm on the maintenance object, or they will pass and remove the maintenance object from further suspicion. Intensive testing continues as long as any errors are being detected on a maintenance object, whether it is alarmed or not. This testing is done at low priority to avoid interfering with normal system operation.

All alarms lead directly to a specific repair activity such as replacing a circuit pack, changing a power supply, or repairing a station set. Such repair activities have a good chance of fixing the problem and retiring the alarm automatically.

4.1.4 System recovery and reconfiguration

The major recovery and reconfiguration strategy is to take selected pieces of equipment out of service (maintenance busy) to preserve good service for the remaining equipment. Faulty trunks and service circuits are removed from service, but care is taken to avoid disabling all the trunks in a trunk group or all the service circuits. Stations are not removed from service except when they have totally failed and cannot provide any service or when the failure mode may disrupt other parts of the system. If any failure prevents the system from processing

calls, emergency transfer stations are connected directly to trunks. Loss of ac power, continuous rebooting, and complete loss of the time division bus are examples of conditions that invoke emergency transfer.

All systems are provided with battery backup for the entire system. If the power outage is short (up to 10 seconds) the batteries will support the entire system. If the power outage is longer than 10 seconds, the system is put into a standby mode where battery power is supplied only to the control complex. All calls are dropped and emergency transfer is invoked. The batteries can power the memory for an additional ten minutes. This shortens the time required to bring the system back to an operational state since it takes several minutes to reload the memory from the backup tape.

4.1.5 System repair

The particular type of repair activity used depends upon which type of maintenance object has failed. Failures in circuit packs connected to the time division bus, including station, trunk, and service circuit packs, are easiest to repair. They can be replaced with minimum impact on the system since it is not necessary to turn off power. The faulty board, indicated by a red LED, is just unplugged and replaced; only those calls using the board are affected. Two additional LEDs are used to guide this repair activity. A yellow LED shows the pack is in use and a green LED shows the pack is under test.

Repair of control circuit packs requires system power down, board replacement, and system reboot. Station repair activities include locating the trouble, wiring repair, and station replacement. Trunk repair activities include verifying the trouble, fixing trunk wiring, and reporting troubles on connecting switching systems.

4.2 Processor complex

The critical boards required to provide system service are the processor, memory, and interface to the switching network.[6]

4.2.1 Processor board

A sanity or watchdog timer, which must be reset periodically, monitors major failures of the processor. If the software fails to attend to this function, the processor is reset and the maintenance circuit pack, an independent processor, is notified. Major system outages are covered in Section 6.2.

Several functions of the processor adjunct circuits are periodically tested, including software interrupts (such as divide by zero), memory management operations, privileged instruction detection, bus time-out exceptions, and initialization and bootstrap ROM checksums.

4.2.2 Memory

The memory has full multibit error detection and single-bit error correction circuitry. It can be reconfigured to run even if every memory access detects a single-bit error. The advantage of this feature is that single failures in the memory do not result in any increase in system outage.

Full, destructive memory tests are run on system initialization (before reboot), including a horizontal and vertical partition test and stuck-at-zero/stuck-at-one test. The following tests are run during system operation:

1. Read all memory test. Read every word in memory to correct soft errors and to force an error report.
2. RAM checksum test. Check that the text segments of all processes have not been modified.
3. Memory error correction test. Check that the error correction and detection circuitry is working and generating the correct failure reports.

If several uncorrectable errors are detected, the system initiates a reboot, memory is fully tested, and the corruption is corrected.

4.2.3 Network control

The network control circuit pack includes both an interface to supervise the port boards and four interfaces for data communication with peripheral devices. These are referred to as the control channel and data channels, respectively.

The control channel is monitored using several in-line error-detection mechanisms, since it is a critical link in switch operations, with all messages to and from the port boards flowing over it. Error-detection hardware immediately reports failures of the Time Division Multiplexed (TDM)* bus clocks. The control channel protocol provides error detection using sequence numbers, checksums and acknowledgments, and error correction using retransmissions. The control channel driver checks the consistency of the interface and maintains a sanity handshake. The control channel interface processor also checks its own internal RAM and ROM for faults during operation.

Several periodic tests check operation of the control channel. The control channel test verifies communications with several port boards that are installed in all systems. The control-channel processor loop-around test verifies operation between the main processor and the control channel processor. Finally, for severe problems, the reset test is used to reset the control channel processor and trigger local initialization tests without affecting the port boards.

* Acronyms and abbreviations used in the text are listed at the back of the *Journal.*

The data channels are similar to other port boards, except that the data side of the port talks to the processor instead of a terminal. Tests include a loopback through one channel, a data loopback between channels, and a dual-port RAM test of the processor interface. When problems escalate, a data channel processor reset is forced.

4.3 Port boards

Some maintenance features apply to all port boards. Except the tone/clock board, all can be replaced with the system powered and running, and only those calls using the replaced board will be disrupted. The system automatically detects the removal and insertion of port boards. Inserted boards are put into service and tests are run. Removed boards are taken out of service and alarmed.

The first priority for port board tests is to protect service. All port tests, including trunks, stations and service circuits, will not run if the port is busy. In addition, all tests will be aborted if an attempt is made to seize the port externally (off-hook on a station or seizure on a trunk) or if a call attempts to terminate on a port (someone calls that trunk or station). If tests must be run on a busy port, the port can be busied out using the system access terminal to force it idle.

4.3.1 Common port maintenance

The port board processor, bus buffers, and control channel interface are common to all port boards. When a port board processor is initializing, it runs a test of its RAM, ROM, and I/O devices and stops if there is any failure. During normal operation many of these tests continue and report in-line errors. The main processor tests the control channel to each board and audits the network connectivity.

4.3.2 Trunks

The tests listed below are run on most trunk types.

1. Seizure test. On trunks that are outgoing, the trunk is seized to verify the trunk signaling and provide good assurance of a working trunk. This tests dial tone reception on those trunks that provide dial tone.
2. Signaling diagnostic tests. Usually the trunk port board processor can exercise signaling mechanisms and detect correct operation of such items as ground detectors, ring detectors and battery feed. In many cases, special circuitry is used to simulate incoming seizures.
3. Tone loop-arounds. This tests that the trunk is able to transmit voice information successfully in both directions. The trunk is put into a loop-around mode, tones are sent, and the returned signal is tested for level and noise.

4.3.3 Stations

The tests listed below are run on stations.

1. Signaling channel test. Digital stations and multibutton stations have a data channel that transmits control information to and from the station. This channel is tested for correct and reliable data.
2. Station present and overcurrent tests. These tests determine if a station is present and test for shorted or open wiring by measuring battery current flow.
3. Tone loop-arounds. These tests are made in the same way for stations as for trunks (see Section 4.3.2).

4.3.4 Service circuits

Service circuits are tested by hooking two circuits together and testing them against each other. For example, tone generators are connected to tone detectors and both are tested at once. Another example is connecting two pooled modems back to back so that the data are converted from digital to analog and back to digital. The faulty service circuit is identified by testing against known good service circuits.

4.4 Miscellaneous components

4.4.1 Maintenance circuit pack

The maintenance circuit pack is periodically tested using a sanity handshake test and tests of the alarm-origination circuitry. If the handshake test fails, a dual-port RAM test is run to test the interface, and the maintenance circuit pack is reset. During reset, the maintenance circuit pack executes a complete set of initialization tests.

4.4.2 Tape

Most error detection for the tape subsystem occurs during normal use. Periodically the processor interrogates the tape subsystem for status and runs data loopbacks through the interface boards. Every 24 hours the tape subsystem self-diagnostics are started and read/write tests on unused tape areas are run. If errors are encountered, a test reads the entire tape and checks the consistency of data stored on the tape. Finally, warning alarms are raised if the tape is not installed or is write-protected.

4.4.3 Power supplies

The distributed power supplies in each carrier are monitored via the maintenance circuit pack, and all but the control carrier power supply can be recycled or shut down. A carrier supply is recycled if it fails; if this does not restore operation, alarms are raised. The battery charger monitors itself and the batteries for problems. Problems such as a

nonworking charger or an open cell in the batteries can be detected. In addition, worn out batteries that no longer hold a charge are detected if the charger does not shift from a high-charge rate to a low-charge rate within the time normally required to charge the batteries.

4.4.4 Environment

Temperature sensors in the cabinet monitor high temperature and lack of airflow. Alarms are raised when these conditions are detected. Airflow monitoring directly detects dirty air filters, eliminating the need for periodic inspection.

V. SOFTWARE MAINTENANCE

In a computer-based system, most features are provided using software. Software maintenance becomes as important as hardware maintenance, since the system can lose functionality when the software malfunctions. This section describes the principles and techniques used in maintaining the software.

5.1 Software maintenance principles

Software failures are fundamentally different from hardware failures. First, reliability of a software component does not change with use; it cannot break. Bugs are discovered and removed over time; but before the bug is found and fixed, the system must be able to recover and operate with the existing software package. Secondly, a bug in a software package does not necessarily prevent the software from performing its task. Only when the bug is exercised by a specific combination of events can the bug cause failure.

Software may also fail if the hardware executing the software has an intermittent error. Intermittent hardware errors may occur if a particular device has low noise margins or if the hardware deteriorates through aging effects. These intermittent errors may have symptoms similar to software bugs.

Although restarting software execution often permits the system to recover from a software failure, this should be done infrequently and with the least possible service disruption. The system has several levels of recovery to deal with software errors, including single-process restart, system-warm restart, system-cold restart and system reboot. More drastic recovery actions are more likely to be effective but are also more service disrupting. This leads to the following strategy: When the maintenance subsystem detects a system failure without knowing the specific reason, it tries a recovery action that has less impact on the system services. Only if that does not cure the problem is more drastic and service-disruptive recovery action invoked. If the cause of a system failure is known, the right recovery level is invoked

immediately, avoiding the risk of the escalation scheme slowing down system recovery.

5.2 Data audit

Auditing is a technique to ensure that system resources have internally consistent states and to check data consistency periodically. Errors detected by data audits are used to start automatic recovery and provide clues to the system developers for debugging purposes. System resource inventories (e.g., message buffers, path descriptors, etc.) are also audited to detect lost resources. Hardwares status (e.g., circuit port status) is audited against the data kept by software to detect any state inconsistency.

5.2.1 Audit philosophy

The fundamental audit philosophy is to find lost resources before the system performance is adversely affected. When an error is detected, the data are restored to a safe system state, one that has the highest probability of not denying service. For example, a trunk or line found in an erroneous state is restored to the idle state. Another basic principle is that if one error is found, all actions taken to restore that resource do not use any other status data associated with that resource because they may be incorrect or inconsistent.

5.2.2 Audit types

5.2.2.1 Data relation audit. Many audits have been developed to check the consistency of internal system status. For example, an audit checks the user station list stored in the user manager process. If a station is shown as on a call, then the service dispatcher process is checked to see if that user is still active on the call. If not, then the user station is marked idle by the audit. Another audit checks the call record list stored in the service dispatcher process. If there is no user on that call, the call record is released to terminate the stranded call record.

Another example is the audit of the touch-tone receiver status which is kept in the connection manager. The audit will check a call to see if an allocated receiver is still active. If not, the audit will release the receiver.

5.2.2.2 Defensive programming. The data used by a process may be obtained from other processes. To prevent erroneous data from propagating through the system, processes typically check input data before using it. Sometimes a process will check for abnormal local data. However, defensive programming must be used wisely so that it does not introduce significant real-time overhead.

5.2.2.3 Process sanity audit. The hardware sanity timer monitors the sanity of the operating system, which monitors the sanity of the

maintenance control process, which, in turn, monitors all other application real-time processes.

5.2.3 Audit control

Data audits are programmed as a set of functions, and each one can be invoked independently of other system operations. When an application process is restarted, it will invoke the path index audit in addition to other process-specific data audits. A central maintenance control process regularly invokes the data audit functions on a time-available basis.

VI. MAINTENANCE CIRCUIT PACK

One circuit pack in the system is devoted strictly to maintenance functions. As long as the processor is running correctly, the maintenance circuit pack serves as a peripheral device for the processor (see Fig. 2a). In this mode, the maintenance circuit pack is used to provide maintenance access to the system and auxiliary maintenance functions. It is not responsible for testing any pieces of the system on an ongoing basis. However, if a fault makes the processor complex unable to run programs effectively, the maintenance circuit pack will assume overall control and intervene to attempt recovery (see Fig. 2b).

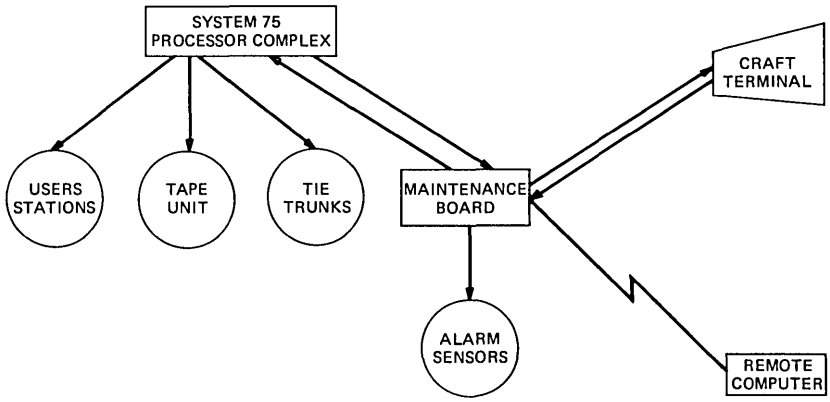
6.1 Normal operating mode

Normally the main processor has responsibility for overall maintenance. The maintenance circuit pack provides the interface to several maintenance functions, including monitoring the status of the ac power, power supplies, battery, and charger. It passes the status of this equipment to the main processor and controls this equipment under the direction of the main processor. It also provides an interface to cabinet temperature and airflow information, external alarms, and system control panel lamps.

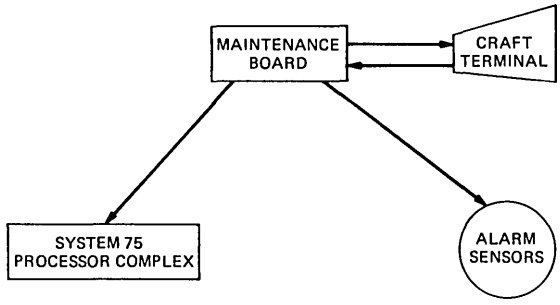
6.2 Operation with a critical system fault

The maintenance circuit pack monitors the operation of the main processor, taking control when the main processor becomes completely inoperative (unable to load or run Oryx/Pecos code). To prevent this control of the system from occurring prematurely, the maintenance circuit pack must see several failures over a period of time before it will assume control.

Once it has assumed control, there are some basic changes and simplifications to the maintenance strategy. The maintenance circuit pack will periodically attempt to restart the main processor. It will automatically place a telephone call and alert the Operations Support Center computer that a major problem exists. It will put the system



(a)



(b)

Fig. 2—Maintenance board connection (a) in normal mode and (b) in stand-alone mode.

into an emergency transfer mode where some telephones are connected directly to the central office trunks to provide basic telephone service. And the maintenance circuit pack will continue to directly monitor and control and most critical parts of the power system.

The maintenance circuit pack provides a terminal interface to help in isolating and fixing system problems. This is particularly important since symptoms of total failure are often the most difficult to diagnose. This interface is a proper subset of the commands used when the system is fully operational, eliminating the need to learn another command language. The commands include displaying the cause of alarms and testing critical circuit packs.

Once the cause of the failure is eliminated, through manual repair of a faulty circuit pack or because some temporarily disrupting con-

dition has stopped, control returns to the main processor and normal operations resume.

VII. MAINTENANCE SOFTWARE ARCHITECTURE

7.1 Overview

The maintenance subsystem is composed of three sections. The first section provides system testing and recovery activities, the second stores and retrieves error and alarm logs, and the last section provides an interface for the technician to request actions and query the status of the maintenance subsystem. The relationship of the maintenance processes is shown in Fig. 3.

A central control process called High-Level Maintenance Manager (HMM) provides a mechanism for the whole system to use in reporting errors. The HMM begins maintenance activities based on in-line error reports, technician commands, or time (periodic maintenance). The time-consuming testing and recovery functions are performed in several different instances of Maintenance Action Processes (MAP). The Maintenance Data Manager (MDM) controls the error and alarm log, condenses error and alarm records, and services queries of the log. The technican can issue maintenance requests through the Maintenance Command Process (MCP).

7.2 Process description

7.2.1 High-level maintenance manager

The HMM is responsible for the strategy of maintenance object operations. The HMM will send reported errors to the MDM process

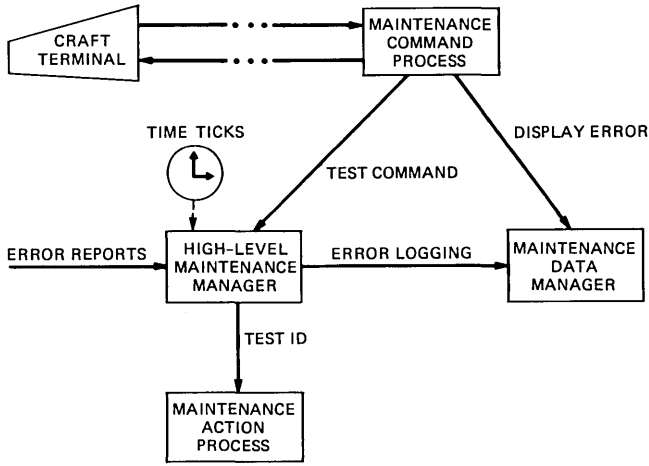


Fig. 3—Maintenance software architecture.

for logging and invoke proper recovery actions. Maintenance can be active and at different states of completion on many maintenance objects simultaneously. For this reason, the HMM multitasks the execution of maintenance strategies.

Errors of all types and severities, some demanding immediate attention, are reported to the HMM. The particular maintenance object strategy determines the priority of the resultant action by dispatching the testing and recovery tasks to a MAP of the appropriate priority. The HMM manages the various MAPs and allocates a MAP when requested by the maintenance object's strategy. Since the MAPs are valuable maintenance resources, the HMM has to control the maintenance load created by background testing to ensure the availability of a MAP should quick testing or recovery response be necessary.

7.2.2 Maintenance action process

There is a set of maintenance test and recovery routines designed for every hardware or software maintenance object type. These are grouped into several different kinds of MAPs, some with multiple instances to provide the required level of concurrency.

1. Initialization MAP (INITMAP). Runs all the critical test and recovery routines, including system initialization and recovery, system power failure handling, and software process recovery.
2. Hardware MAP (HWMAP). Tests for most of the hardware maintenance objects, including processor, port circuit packs, trunks, stations and the tape drive, are grouped into the hardware MAP. Instances of this MAP are created at different priorities to provide the required responsiveness.
3. Data Audit MAP (AUDIT). All the data relation audits are grouped into one MAP.

7.2.3 Maintenance data manager

The MDM is responsible for the error and alarm log. It receives error reports and requests to raise or resolve alarms, and maintains them in an internal database. It supports queries of this database for the maintenance command interface. When alarms are raised or resolved, the MDM controls the proper alarm indications.

7.2.4 Maintenance command process

The MCP converts external maintenance command requests into internal test requests and queries. For example, a request to test a trunk group is expanded into a sequence of tests on the members of the group.

7.3 Message flow examples

The following examples help explain the operation of the maintenance subsystem.

7.3.1 Maintenance command

This example covers a technician test request on a station. (Refer to Fig. 4.)

The technician's command originates in the administration subsystem, which is responsible for the forms-based human interface.[7] The command arguments are passed to the MCP, representing the request to run a test sequence on a specific extension. The MCP converts the station extension to a maintenance object identifier and initiates the tests through the HMM. The maintenance object strategy in the HMM requests a MAP to execute the tests. The MAP executes the test and sends the results back to the HMM, where they are forwarded back to the MCP. The MCP formats the output for display by the administration subsystem.

7.3.2 Digital station failure

This example covers the failure of a digital station leading to an alarm. (Refer to Fig. 5.)

Typically a port failure will appear as an in-line error detected and reported by the port board firmware. In this case a data link error is reported by the digital station board. This error message is routed to the HMM, where a threshold counter is incremented for this mainte-

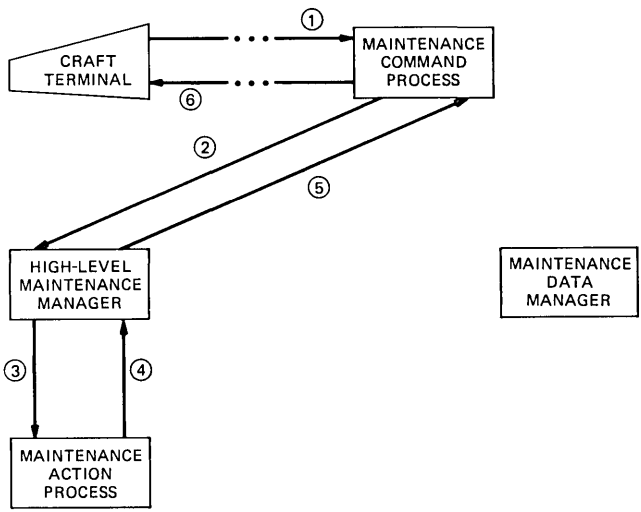


Fig. 4—Craft command message flow.

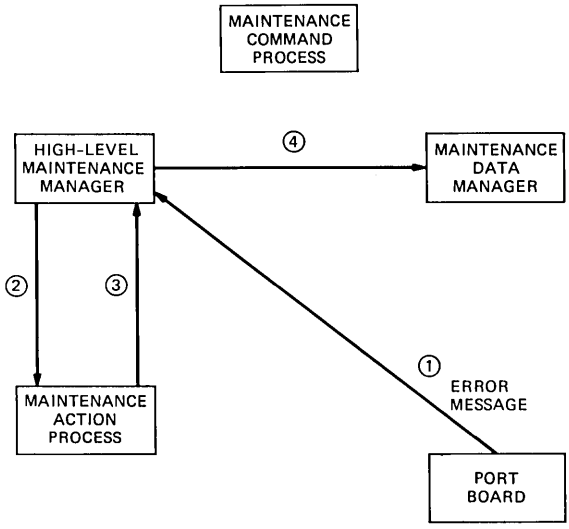


Fig. 5—Station link failure.

nance object. The counter being active causes the maintenance object's strategy to dispatch a test to verify the error.

One test to be run, based on the active threshold counter, is a data link loop-around. This test fails since the data link is faulty and causes the counter to be incremented again. The object strategy schedules more frequent testing while troubles are active on that port. With the test failing, the counter soon exceeds its threshold and starts the MAP alarming action to raise the alarm with the MDM.

VIII. PERSON/MACHINE MAINTENANCE INTERFACE

A CRT terminal was selected to be the system access terminal since this provides the best interface for repair personnel. The terminal provides a powerful and flexible human interface, with a simple command language that is easy to learn and remember. The same terminal and command interface is used for maintenance and administration.[7] Some examples best serve to illustrate the ease with which maintenance activities can be requested.

The simple English-like command language can be used to request tests of specific parts of the system. A circuit pack can be tested by entering:

```
test board A13
```

where `A13` is the location of the circuit pack in the cabinet. Station extension 235 is tested by the command:

```
test station 235.
```

The need to remember and call for specific tests has been eliminated. Each hardware maintenance object has a short test and a long test, with the short test as the default. The short tests are not service disruptive and complete quickly. The long test causes all tests to be run on a maintenance object, it takes longer to complete, and is more likely to disrupt service. This complete test is specified by adding the word *long* to the test command. For example:

```
test tape long
```

The interface also makes use of the terminal forms capability. The system has many options that allow the selective display of errors, restricting it to errors on specific pieces of equipment, to errors occurring over a specific interval of time, or to certain types of errors. Rather than complicating the command that displays errors and making it difficult to remember, the simple command:

```
display errors
```

is used, which brings up a form on the CRT screen. Then any display restrictions are made using form entries.

IX. REMOTE MAINTENANCE CAPABILITIES

System 75 has a complete set of remote maintenance capabilities. These capabilities are provided by a dial-up connection to AT&T Information Systems Operations Support Center centralized maintenance computer.[5] Communication between the system and the support center computer can be established in either direction using the X.25 protocol. This protocol provides error detection and retransmission and ensures that error-free information is exchanged between the two computers.

All the components required for this interface are part of the maintenance circuit pack, including a 212-compatible data set operating at 1200 baud, a line interface, and an automatic dialer. This integrated design speeds installation since it is only necessary to connect two wires from the central office.

From a maintenance point of view, one of the most important capabilities of this remote access is automatic alarming. Whenever an alarm is raised, the system will dial up the Operations Support Center computer and report the details.

The Operations Support Center computer allows remote execution of all maintenance commands that are supported by the system administration terminal. Commands can be issued from the support center regardless of which end made the call, allowing trouble analysis to begin immediately after a trouble is reported.

Other capabilities of the Operations Support Center computer include up-loading and down-loading of translations and remote program update. Any changes to the load module are transmitted and added to a separate patch area on the tape. A remote command then reboots the system, updating the program without a visit to the site.

These remote maintenance functions provide an advantage over strictly on-site maintenance since the remote site can respond more quickly and visits to the site can be more carefully planned. This results in better, faster service at lower cost.

X. SUMMARY

The major objectives in the maintenance design were to make the system reliable, self-testing, and easy to repair. Feedback from the field has been very positive. Maintenance operations, including the terminal interface, alarming methods, error logging, and automatic testing have been successful in providing low-cost support. Most changes based on field experience have been fine tuning maintenance strategies to cover unexpected failure modes or adjusting the sensitivity of alarming thresholds. Software maintenance capabilities have allowed us to weather early software faults and will continue to provide insurance against undiscovered problems.

XI. ACKNOWLEDGMENTS

The design described in this paper is the culmination of the ideas of many people at AT&T Information Systems Laboratories. We would like to acknowledge the efforts of all these people, with particular acknowledgment to M. C. Wei for his contributions to the maintenance software design.

REFERENCES

1. E. J. Braun, "Maintaining the DIMENSION® 400 PBX," Bell Lab. Rec., October 1976, pp. 244–8.
2. P. W. Bowman et al., "1A Processor: Maintenance Software," B.S.T.J., *56,* No. 2 (February 1977), pp. 255–87.
3. H. J. Beuscher, "No. 5 ESS Maintenance Software," IEEE Trans. Commun., *COM-30,* No. 6 (June 1982), pp. 1386–92.
4. K. T. Fong, G. R., Sager, and J. A. Melber, "System 75: ORYX/PECOS Operating System," AT&T Tech. J., this issue.
5. J. E. Smathers and N. T. Tsao-Wu, "RMATS—Remote Maintenance System for DIMENSION® PBXs," Conference Record—Int. Conf. on Communications, June 1979.
6. L. A. Baxter et al., "System 75: Communications and Control Architecture," AT&T Tech J., this issue.
7. H. K. Woodland, G. A. Reisner, and A. S. Melamed, "System 75: System Management," AT&T Tech. J., this issue.

AUTHORS

Kang-sen Lu, B.S. (Physics), 1974, National Tsing-Hua University; M.S. (Computer Science), 1976, National Chiao-Tung University; Ph.D. (Computer Science), 1981, University of Pennsylvania; AT&T Bell Laboratories, 1981–1982; AT&T Information Systems Laboratories, 1983—. Mr. Lu developed the maintenance control software and system recovery strategy used in the System 75 release 1. His current responsibility is on the maintenance software architecture of the System 75 release 2.

John D. Price, B.S. (Electrical Engineering) 1977, Cornell University; M.S. (Computer Engineering), 1978, Carnegie-Mellon University; Bell Laboratories, 1977–1982; AT&T Information Systems Laboratories, 1983–1984; AT&T Consumer Products, 1984—. Recently Mr. Price has worked on maintenance software and product delivery for System 75. He is currently a Supervisor of a group designing residential terminals. Member ACM, IEEE.

Thomas L. Smith, B.S. (Electrical Engineering), 1967 and M.S. (Electrical Engineering), 1969, The Massachusetts Institute of Technology; Bell Laboratories, 1969–1982; AT&T Information Systems Laboratories, 1983—. Mr. Smith has been involved in the design and systems engineering of maintenance features for several systems including 3 *ESS,*™ No. *5ESS*™ and minicomputer maintenance support systems. He is currently the Supervisor of System 75 maintenance and firmware group. Member Eta Kappa Nu, Tau Beta Pi.

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

The Oryx/Pecos Operating System

By G. R. SAGER,* J. A. MELBER,† and K. T. FONG†

(Manuscript received July 11, 1984)

The System 75 Office Communication System is the first field application of the Oryx/Pecos operating system, a message-based system which supports real-time, distributed applications. Its interprocess communications mechanisms provide a structuring tool similar to monitors, capabilities, and abstract data types. This paper describes the principal concepts implemented in the operating system kernel, and presents the essential system processes. Support for application design techniques is discussed and related to proven software engineering principles, including information hiding and modularity. Specific examples are drawn from System 75 for call processing and maintenance.

I. INTRODUCTION

The Oryx/Pecos operating system provides an environment for real-time, distributed applications. By the term "real time" we mean that the performance of the system is reasonably fast and, above all, easy to predict. By the term "distributed" we mean that assignment of elements of the application to processors can be made apparent or transparent, as befits requirements.

This operating system is intended to extend the applications implementation language to include powerful structuring tools—similar to

* Currently with Sun Microsystems, Mountain View, California. † AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc.

monitors,[1] capabilities,[2] and abstract data types[3]—as active elements of the application.

System 75 is the first Oryx/Pecos field application. In this initial application, the operating system runs on a single *Intel** 8086-based processor specially designed for System 75. All of the System 75 call processing, maintenance, and administration software is built on top of this operating system. To provide a responsive, feature rich, and extensible Private Branch Exchange (PBX)† with maximum system capacity, System 75 requires an operating system that is fast and can provide predictable performance. Use of other operating systems, such as the *UNIX*™ operating system, would not have met these needs.

The Oryx/Pecos operating system is implemented as a kernel (Oryx) and a set of essential system processes (Pecos).

II. THE ORYX KERNEL

The kernel (or "nucleus") of an operating system is a basic set of primitive operations from which the remainder of the system is constructed. The Oryx kernel appears to the programmer as an instruction set that manipulates *processes*, *messages*, and *paths*.

A *process* is the independent, sequential execution of a program. Processes may share instruction (I) space, but each process has its own stack and data (D) space. D spaces are not shared. The separation of D spaces is enforced by a memory-management device. Communication between processes is limited to messages and data transfers (as discussed below).

Processes are the source of asynchrony; the progress of independent but similar computations is modeled by processes with the same I space, each at a different stage of execution. Processes can act as monitors to solve critical sections and to enforce system policies.[4-6] Much of the operating system itself is contained in processes built on top of the kernel. Building with processes avoids a monolithic structure by enforcing physical separation of the system components. Parts of the operating system may be changed without affecting other parts (assuming interfaces are preserved). Bugs tend to be isolated. The system can be configured by adding or deleting system or application processes. Parts of the system or application may be designed to fail and recover without widespread repercussions.

A *message* is a small, fixed amount of information transmitted from a source process to a destination process. The source and destination processes may be on the same processor or on different processors.

---

* Trademark of Intel Corporation.

† Acronyms and abbreviations used in the text are defined at the back of the *Journal*.

Messages are also used to transfer arguments and results between the kernel and processes. Message transmission and reception simplifies interfaces and allows arguments and results to pass between processors for process-kernel interactions and for kernel-kernel interactions as they do for process-process interactions. In this respect, the kernel bears many similarities to a process. Similarly, device drivers are implemented as part of the kernel and interface with processes via messages.

Messages provide synchronization: they request an action or indicate that a requested action is complete. Messages are small (16 bytes) to reduce time for copying and space for buffering. Messages are fixed in size to avoid fragmentation in buffer allocation and disagreements on size between the source and destination.

Processor allocation (*dispatching*) is controlled by a combination of process priority and message transmission. When a message is sent from a lower- to a higher-priority process, the processor is allocated to the higher-priority process. This, coupled with nearly constant message transmission times, allows for greater ease in performance analysis of critical message sequences. Execution of a process is sequential; blocking occurs only as a result of waiting for a message and unblocking only as a result of message arrival. Message reception is entirely voluntary; the queueing of messages and ability to selectively receive on an "or" condition eliminates the need for signals or events and keeps the execution of processes sequential. The kernel translates device interrupts into messages and delivers them to device-controller processes. If an interrupt unblocks a higher-priority process, the currently executing process is preempted and will resume execution when all higher-priority processes have blocked.

Paths are a protection mechanism patterned after "links".[7] Message flow over a path is unidirectional: the *owner* of the path is the source, and the *creator* is the destination. In Fig. 1a, process A owns a path to process B, the path creator. For the path owner, the path represents a capability of sending messages; conversely, for the path creator, the path represents an agreement to receive messages. Paths may cross processor boundaries; this is transparent to both the owner and creator.

Messages can be used to set up new path connectivity. Process A agrees to receive messages from process B by creating a path (Fig. 1b). Process A then *passes* the path to process B over an existing path (Fig. 1c). Passing the path changes ownership, thereby enabling process B to send to process A. The initial paths among processes are controlled by the process manager (as described in Section III).

Paths have features to enhance their usefulness as a protection

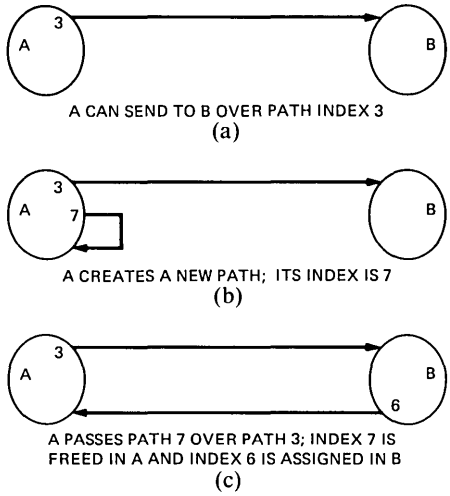


Fig. 1—Passing a path.

mechanism. Features are selected by the creator at path creation time, and they are enforced by the kernel; processes cannot "forge" a feature.

1. The *class* allows the creator to be selective in receiving messages. The creating process can specify one of seven classes, or can have the kernel select a class from a pool of classes not in use by the process. In a receive condition, if no messages are available on paths in the set of specified classes, the process blocks until one arrives. The kernel includes the path class as a part of all messages received over a path.

2. The *tag* allows the creating process to encode information concerning the reason for creating the path. The kernel includes the path tag as part of all messages received over a path to remind the creator of that reason. The tag is often a pointer to a data structure describing the state of the conversation on the path.

3. *Properties* can be used to prevent an owner from duplicating a path, to cause generation of a notification to the creator if the path is destroyed, or to limit path use to the transmission of a single message. The *duplicatable* property allows the owner of a path to make a copy of the path. Notifications enable the creator to keep an account of path sources; whenever an owner explicitly destroys a path with the notification property, the creator receives a notification with a count of how many copies of the path remain. A process death implies that paths owned by the process are destroyed; thus, a notification can inform the creator of the death of an owner. Limiting path use to a single transmission ensures that the creator can expect exactly one message back.

4. *Restrictions* limit the ability of an owner to give paths to the

creator. Restrictions help a creator avoid security problems generally classed as the Trojan Horse or cloying, i.e., false or unwanted paths.[8]

5. *Data transfer* paths and messages can set up an agreement to transfer bulk data in a manner that appears to the user as Direct Memory Access (DMA) input/output. The technique is convenient and more robust than messages for moving arbitrary amounts of data, as it avoids problems of message queue exhaustion and the need to sort the data stream from other incoming messages at the receiving end; furthermore, the creator of a data transfer path is allowed to execute while the data transfer occurs. Local or remote transfer of data is allowed.

Certain combinations of properties and restrictions occur frequently and are described below for convenience of discussion:

request—allows for the passing of *reply* paths and for the acquisition of *resource* paths. *Request* paths can be duplicatable.

resource—notifies the creator if the path is destroyed. It allows passing of *reply* paths and abstracts the allocation of a resource to the owner from the creator. *Resource* paths can be duplicatable.

reply—destroyed on use or notifies the creator if the path is destroyed. It allows for the passing of *resource* paths. *Reply* paths cannot be duplicated.

As an example of how to use processes, paths, and messages, we consider how one might implement a File System Server (FSS). In Fig. 2, process A is a client of the FSS; it may or may not be on the same processor as the FSS. Process A has the capability of opening

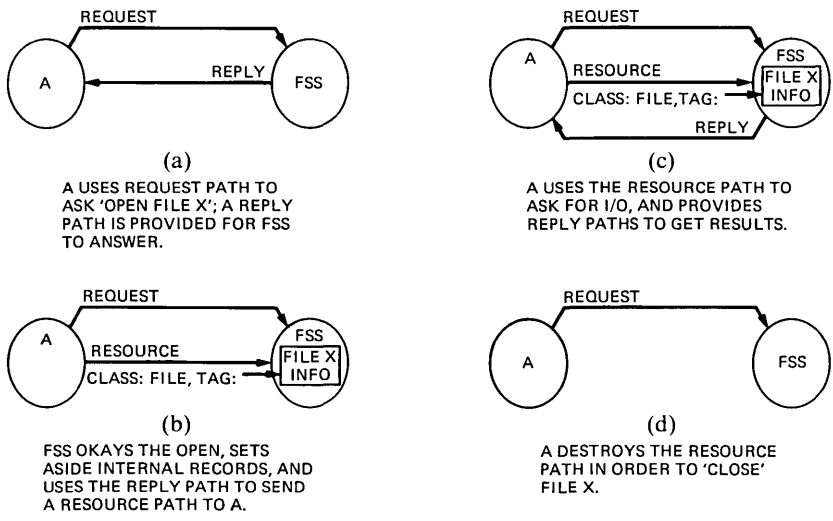


Fig. 2—File server example.

files; this capability is represented by the request path from A to the FSS, as in Fig. 2a. Process A opens a file by sending a message over the request path that says *open file x* and providing a reply path for a result (Fig. 2a). The FSS acknowledges the request by creating a resource path and using the reply path to pass it back to A (Fig. 2b); the FSS assigns the class it has reserved to represent open files and chooses the tag to point to a data structure containing data pertinent to the state of file x. The resource path represents a capability for A to ask the FSS to operate on file x. To operate on file x, A sends a message over the resource path and provides a reply path if a result is required (Fig. 2c). Note that the FSS relies (1) on the class to indicate that the message concerns an open file, and (2) on the tag to determine that the file to be operated on is x. Thus, it is impossible for A to lie (purposely or accidentally) to the FSS concerning the identity of the file to be operated on. Furthermore, A can pass only reply paths to the FSS so the FSS divests itself of paths passed from A when it sends a result. Process A closes file x by destroying the resource path (Fig. 2d). Note that closing the file occurs as a result of a destruction notification, rather than a message sent by A. Destruction assures the FSS that A can no longer communicate regarding file x, and it guarantees that the file is closed if A dies without explicitly closing.

This scenario resembles the control of files in the *UNIX* operating system (see Figs. 3 and 4). Process A's path (file) table is kept in protected space and managed by the kernel; paths (files) are referenced by an index into the path (file) table, which contains sensitive descriptor information. Automatic clean-up is possible because the information is kept in a disciplined fashion by the kernel, rather than in an undisciplined fashion by the process itself. *UNIX* system processes

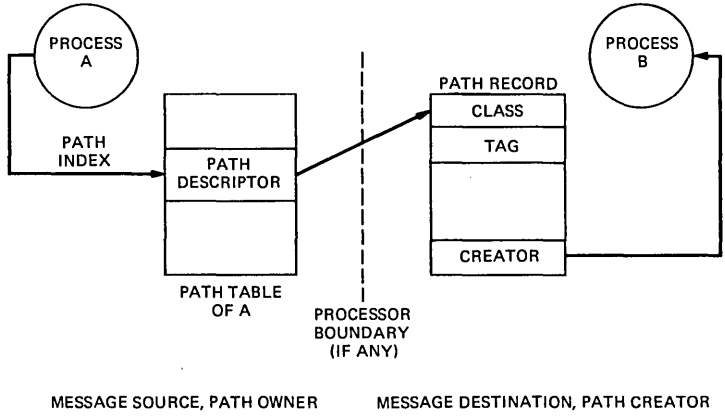


Fig. 3—The Oryx implementation of paths.

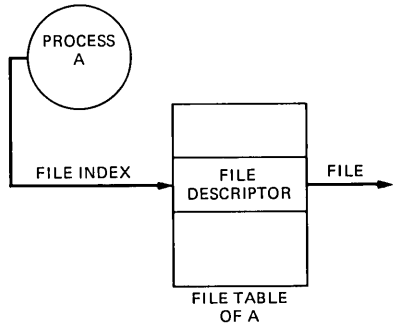


Fig. 4—The *UNIX* system implementation of files.

often use files as an abstraction mechanism to control resources. Paths are a more basic device to serve the same end in a distributed system.

It is important to note that the Oryx kernel separates the concepts of *mechanism* and *policy*. Paths provide a basic protection mechanism which processes can use to enforce protection policies. Thus, it is possible for process implementors to set and enforce policies according to their own requirements. This is important because the System 75 applications resemble an operating system in that they are concerned with the allocation and management of a complex set of resources; in many operating systems, these tasks can be accomplished efficiently only by implementing parts of the application in the operating system itself. Separation of policy and mechanism decoupled implementation of the System 75 applications from that of the operating system by reducing the need for the application developers to modify the kernel or operating system processes.

III. THE PECOS SYSTEM PROCESSES

The kernel relies on a set of system processes to provide certain maintenance and policy-making functions. In a distributed system, both the kernel and essential system processes are replicated on a per-processor basis.

The *phantom* acts on behalf of dead processes. When a process dies, processes owning paths that the dead process created execute asynchronously and may transmit messages to the dead process before learning of its death. In this case, the kernel delivers the messages to the phantom process for cleanup. The phantom destroys passed paths in the messages, thereby propagating destruction notifications to the creators of the passed paths.

The *Leisure Time Manager* (LTM) simplifies the kernel's dispatching decisions by ensuring that there is always a process to execute.

Once the LTM is allocated the processor, no other processes will execute until an interrupt occurs. The LTM runs at the next to lowest priority. Processes set to a lower priority than the LTM never execute, but otherwise appear to be normal processes. This is useful for debugging; a process being debugged can be halted without affecting the rest of a running system by placing it at the lowest priority.

The *Process Manager* (PM) is the first process to execute; it creates all other processes. The kernel defers all process creation and destruction decisions to the PM. The PM creates processes by associating memory-resident I spaces with the D spaces it allocates dynamically; processes remain in memory until they die. The PM determines the set of "standard paths" for a new process. Standard paths play a role similar to `stdin`, `stdout` and `stderr` in the *UNIX* system: a new process knows that some path indices can be used to obtain standard system services.

The *Network Manager* (NM) is created early during initialization, and all subsequent processes are created with a path to the NM. The NM allows processes to "supply" it paths with associated symbolic names and to ask for copies of paths by those symbolic names. This *name server* function allows processes to obtain paths without relying on ancestral relationships. The NM also provides a gateway to the NMs on other processors, and the NM is therefore useful for establishing contact with services on those processors. In the case of a processor failure, the NMs on functional processors will establish contact with the NM on a recovering processor and will provide a means to recover communications for the other processes in their processors.

Several other system processes are not essential, but are, nonetheless, useful in many applications. The *timer manager* provides alarm clock and time-of-day services. In System 75, the timer manager is used to time calls (for accounting records), to provide feature timing, to provide route timing (i.e., going to coverage), to recognize "no response" situations, and to schedule periodic maintenance activities. The *console manager* provides output to the system console and allows an "operator" to inspect the running system; the *error logger* saves log messages that may be useful for later debugging or accounting; and the *shuffler* provides long-term enforcement of the processor allocation policy. The console manager, shuffler, and error logger are not used in System 75. Finally, *optics* is a system process with special privileges to allow it to act as a debugger for other processes. The role of optics as a debugger is enhanced by the fact that it can track and display use of the kernel by a process. A more detailed description of optics can be found in Ref. 9.

IV. APPLICATION DESIGN

When designing an Oryx/Pecos-based application, a number of structuring techniques can be employed. We will discuss the most important and common ones in this section.

As previously indicated, processes provide *asynchrony* and *modularity*. However, these benefits would be limited without the ability for processes to communicate. Using paths and messages, it is possible to construct applications as a collection of cooperating processes. The principles for decomposing an application into processes are analogous to those used when designing with subroutines, *abstract data types*, or *monitors* (for example, see Ref. 10). The judicious use of paths and path features can enforce design decisions through a mechanism very much like *capabilities*.

When using processes for resource control, it is important to observe a *layering* of responsibility which preserves a *client-server relationship* through the layering. At any given instant of time, only *request* and *resource* paths should point down (from client to server) across layers and only *reply* paths should point up across layers, as in Figs. 5 and 6. When properly applied, this structure can make the application free of deadlock (Ref. 4 contains an excellent discussion of this).

Since processes communicate via paths, rather than directly to another process, it is possible to effectively hide the implementation of a layer consisting of many processes. This technique is used in two ways in System 75: (1) to provide multiple threads of execution and (2) to provide service through cooperation of servers.

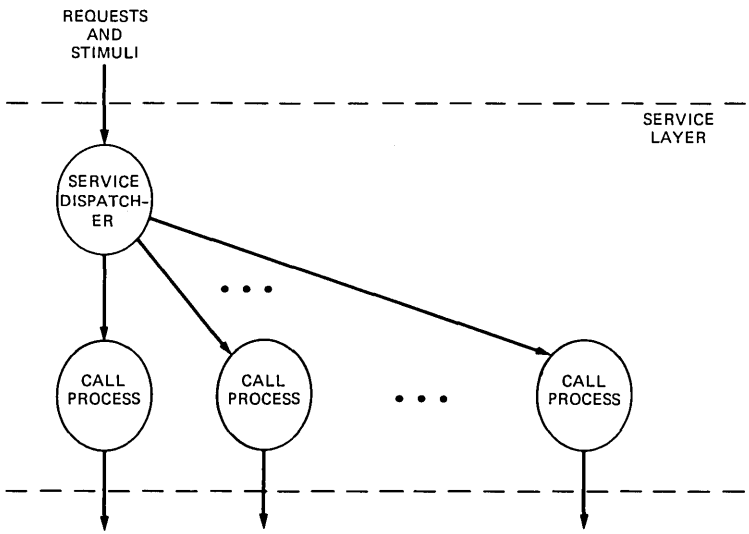


Fig. 5—Multiply threaded layer.

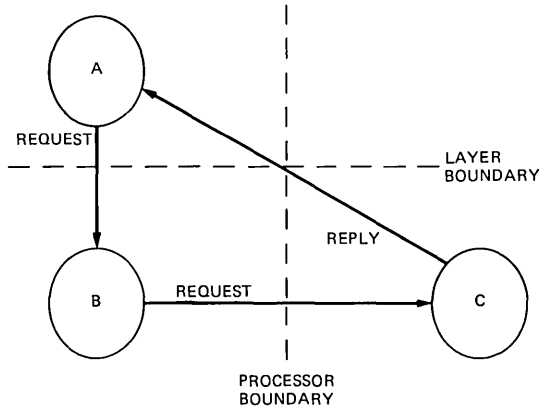


Fig. 6—Cooperative servers.

4.1 Multiple threads of execution

A PBX must handle many simultaneous phone calls, each potentially in a different stage of completion. This could be designed as one large, complex process, but System 75 implements the highest level of call control with two types of processes (Fig. 5): the Call Process (CP) and the Service Dispatcher (SD). There are many CPs sharing the instructions of a program that describes the sequence of steps required to control a phone call. The single SD process receives stimuli related to calls and directs them to the appropriate CP; when necessary, the SD will allocate an available CP to a new call or will make a CP for a disconnected call available. This structure allows asynchronous treatment of many phone calls and simplifies the coding of features by making feature control a conventional sequential programming task. Furthermore, this implementation is transparent to the other layers. A detailed discussion of this structuring technique can be found in Ref. 11. Details on the implementation of the SD and CP processes can be found in Ref. 12.

4.2 Cooperation of servers

In several cases, it is possible to subdivide the responsibilities of a layer to be shared among processes. This is sometimes useful when the coding is divided along similar boundaries. Figure 6 illustrates a typical structure for cooperative servers. It is important to note that when a request is forwarded to a peer server, the forwarding process can divest itself of all further responsibility by forwarding any passed paths with the original request. Thus, in Fig. 6, a message with a reply path is sent from A to B, B forwards the message and reply path to C, and C uses the reply path to respond directly to A when the service is complete. This technique will become more important when layers are

extended across processor boundaries; for example, messaging features (leave word calling, mail, etc.) may provide for messages stored in local memory or in a remote file server. A, of course, has no knowledge of whether or not its request is being handled by another processor.

V. APPLICATION EXAMPLE

To illustrate some of the concepts described above, this section presents a concrete example of how some Oryx/Pecos facilities are used by applications software in System 75. The communications subsystem is a set of processes that allow for direct data terminal communication with the administration and maintenance service processes. It consists of two control drivers, six identical data drivers, up to six identical Terminal Controllers (TCs), and the Communications Manager process (COM). The COM provides resource control for direct data ports and presents a uniform interface to the various service processes. Direct data communications with System 75 processes is possible through two main processor peripheral devices: a maintenance board (providing two data ports), and the data channel part of the switch interface board (providing four data ports). The service processes are generally unaware of what device they are using. Each data port is assigned a data driver, and each board is assigned a control driver. The process and path structure for a service process accessing port 1 of the maintenance board is shown in Fig. 7; the following paragraphs detail how this structure is arranged.

During initialization, the COM exchanges paths with each driver and supplies a path to the NM. All paths from data drivers have the same unique class. The tag of each path from the data drivers specifies the port being handled by that driver. All paths from control drivers have the same unique class. The tag of each path from a control driver specifies the board being controlled by the driver. Thus, when a message arrives, the COM process can determine the sending driver type from the class of the message and can determine which driver of that type from the tag of the message.

Service processes obtain a path to the COM from the NM. This path is used by service processes desiring data port access. The service processes typically send a message (detailing the request) and pass a reply path to the COM. The COM exchanges call progress messages (OFF-HOOK, ON-HOOK, etc.) with a control driver to establish a data call. It then requests creation of a TC process by message interchange with the Process Manager. Paths to the appropriate data driver are duplicated and passed to the newly created TC. Paths to the TC are in turn passed to the requesting service process (over the reply path mentioned earlier).

Thus path connectivity is established between the driver (which

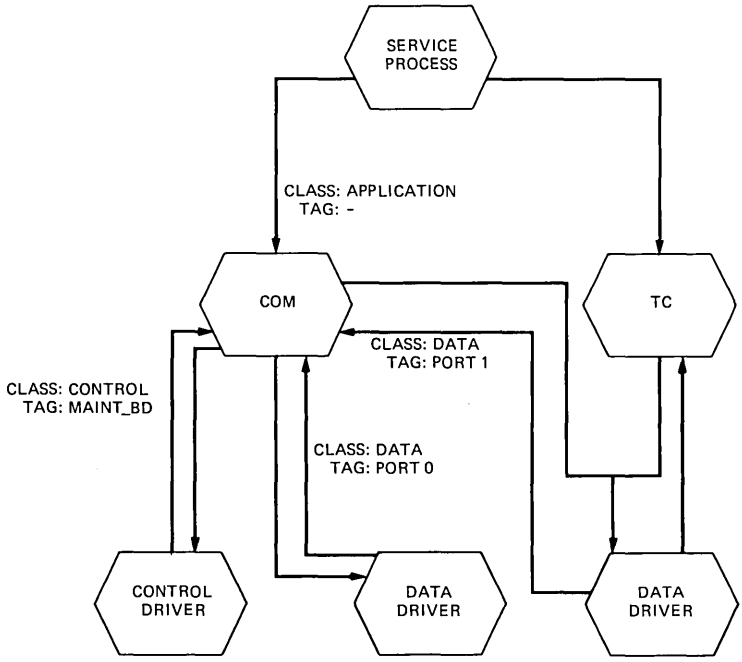


Fig. 7—Communications subsystem processes and paths for a service process with access to port 1 of the maintenance board.

manages the data port hardware), the TC (which provides higher-level data communications functions such as echoing), and the service process (which interprets the user keystrokes).

VI. MAINTENANCE FEATURES

In keeping with the overall system philosophy, the operating system attempts to provide mechanisms by which the application itself can implement a maintenance policy to detect and deal with extraordinary situations. The operating system allows for the provision of a designated application process to define the application maintenance policy; in System 75, this process is called the *High-Level Maintenance Manager* (HMM). If the HMM is not present, the operating system uses default rules to govern its behavior. If the HMM is present, the kernel and/or certain system processes will direct messages to the HMM warning of extraordinary situations and will rely on the HMM to take further action to deal with the situation.

The most basic Oryx/Pecos maintenance function is *process death.* Process death is accomplished when the PM forces a process to execute a program which causes it to release all of its system resources. In many cases, a process which encounters an error situation can correct

the problem by simply committing suicide. Certain processes are considered essential to the well-being of the application, and more drastic actions are required if they die. The PM consults the HMM before forcing the process to execute the death program; if the HMM decides that the process is essential, it will cause more extensive application maintenance to take place. A typical action would be to attempt to restart the process or a group of processes. Restart means that process data are preserved, the process stack is initialized and execution is resumed at the beginning, with arguments to indicate that a restart is taking place; if properly designed, a restarted process may be able to resume normal operation. It is possible to restart some processes after the operating system is reinitialized and thereby avoid the need to reload process instructions or data.

The NM supports asynchronous initialization and recovery by allowing processes to request a copy of a named path whenever it is supplied to the NM. A process which makes use of this feature will receive a fresh copy of the named path whenever it is supplied to the NM, and will therefore be in a position to make use of the latest version of the service provided over the path. Furthermore, the NM keeps copies of all supplied paths and will reissue copies on demand.

An example of the use of process restart and NM facilities is the Board Manager (BM) process. When the BM begins execution, it checks its arguments to determine if it is restarting. If it is, it uses a kernel audit operation to compare the contents of its path table with its internal record of indices of paths it owns. Paths in the path table but not in its records are destroyed; an example of a path which would be destroyed is a reply path whose index was stored in the stack. If its records indicate that a path is missing, a fresh copy of the path is demanded from the NM. When reinitialization is complete, the BM resumes normal operation.

Oryx/Pecos interfaces are provided to allow the HMM to estimate the overall ability of the system to provide service. For example, the kernel notifies the HMM when any available system resource (e.g., message buffers) goes below a low-water mark or above a high-water mark. The HMM uses low-water marks to detect and recover from overload situations; in this case the HMM may lower the priority of some work and/or prevent new work from entering the system. When a high-water mark is passed, the HMM will cause the system to resume normal operation.

Finally, the HMM directs auditing of operating system resources to detect problems in how the application is using those resources. For example, if a process has failed to detect or clean up properly after the death of another process, it may own a path which is no longer valid; in the kernel, this appears as a path descriptor with an invalid pointer

to a path record (see Fig. 3). Errors of this type are logged and clean-up actions are taken.

More details on the operation of the HMM, BM, and other components of System 75 maintenance can be found in Ref. 13.

VII. PERFORMANCE

The Oryx/Pecos operating system is designed to support real-time applications such as call processing. To provide this support, the operating system must be fast, and it must provide facilities so that system performance can be easily predicted.

Performance data are given for an 8-MHz *Intel* 8086 processor running with two memory wait states and equipped with memory management hardware designed specifically for System 75. Measurements were taken using a special I/O device; in-line I/O instructions output 16 bits of data to the device, which time-stamps the data and buffers it for later data reduction. The overhead for this instrumentation is very low and the granularity of the time-stamps is 10 microseconds. The measurements presented include time for kernel entry and exit, argument and result transfer, and process dispatch time, as well as time for the kernel operation itself.

There are variations in the observed message transmission times due to different possible states of the sender and receiver processes at the instant the message is sent. For example, if the sender unblocks the receiver and causes it to execute next, the observed time to transmit a message is:

send	0.62 ms
receive	0.29 ms

for a total of 0.91 ms. If, however, the sender queues up the message and the receiver dequeues it later, the times are:

send	0.50 ms
receive	0.47 ms

for a total of 0.97 ms. In many cases, the total time to transmit a message is the most important factor in performance, rather than the time attributable to either the sender or receiver. As can be seen, the total time for the two extreme cases above shows little variation and could be modeled as a constant for all cases.

When a path is passed with a message, the total time to transmit the message increases to a range of 1.35 to 1.45 ms. For any message sent over a reply path, 0.24 ms must be added to account for the implicit destruction of the path. The time required to create a path is 0.69 ms.

In practice, a typical application scenario is (see Fig. 2c):

1. A client process creates a reply path (0.69 ms),
2. The client sends the reply path with a message asking for service, causing the server to execute (1.35 ms),
3. The server sends a result message back over the reply path, queueing the message (0.97 + 0.24 ms).

The total time for the kernel operations involved is 3.25 ms. The Oryx kernel provides a specially packaged *call* operation to replace the client operations described above, thereby reducing the total time to 2.03 ms.

This type of performance data has been used to create accurate models of System 75 performance; these models were used during the design and implementation phases of System 75 call processing to maximize system capacity.

In addition to fast operation, other factors help make this operating system suitable for real-time applications. Real-time applications such as call processing are characterized by having to guarantee response times to external stimuli at maximum capacity. Thus, the Oryx/Pecos operating system provides facilities for improving response times and for making them predictable. Dispatching facilities (process priorities, priority preemption, and intimate coupling of dispatching and message transmission) enable the application designer to keep the processor allocated to the most critical work. Selective message reception using classes gives the designer the ability to deal with multiple, asynchronous message sources efficiently; this is important because real-time applications often need to respond immediately to stimuli from a number of sources, which may come in any order. Messages may be sent and received either synchronously or asynchronously. Synchronous operation is generally simpler to design and consumes less processor time, while asynchronous operation allows the designer to improve response times by breaking complex operations into short, non-atomic segments of execution and allowing time-critical requests to be handled sooner. Finally, the operating system itself is implemented so that critical operations are fast and "available"; that is, complex operating system functions are broken down into smaller segments so that the less important work can be deferred until later.

VIII. SIZE

The size of the Oryx/Pecos operating system is difficult to characterize in general terms because, to a large extent, its size is determined by the application it services; System 75 gives us a single data point from which to work. Thus, we will limit our discussion to the major factors which determine system size.

The application can directly influence system size in the following ways.

1. Use of services. If an application does not make use of a service, the process which provides that service can be eliminated (for example, the shuffler).

2. Per-process resources. Every application process requires a certain amount of system resources (for example, path tables); the number and size of these resources can be configured at compile time to suit the application.

3. Drivers. Drivers are included in the system size, but the application itself determines which drivers are needed.

Certain implementation decisions in the operating system influence size. In most cases, trade-offs between size and speed were decided in favor of speed, especially if the size came in the form of instructions. In the per-process and per-path data structures, the implementation tends to favor small size. Finally, additional instructions and data to serve the maintenance requirements of System 75 tend to increase the operating system size.

The most important Oryx/Pecos contribution to decreasing the overall size of an application is *shared libraries*. Shared libraries permit the sharing of a single physical copy of utility functions by all system and application processes. For example, any process that needs to format an output string does so by executing the same physical instructions as all other processes. The potential space savings is large: if an application has 25 different instruction spaces for processes and each makes extensive use of a 10K-byte shared library, the savings is slightly less than 240K bytes over what would be required if each process had its own copy of the library functions. The savings is not exactly 240K bytes because each process contains a few instructions to interface to each of the functions in the shared library.

It is not possible to give precise numbers for the space savings due to shared libraries in System 75 because the existence of shared libraries affects the implementation strategy. Rather than minimizing the size of library functions, and perhaps tailoring variants for individual processes, it is more advantageous to increase the generality of the functions to make sure that the maximum number of processes can make use of them. Furthermore, with the decreased size of the multiplier, there is a greater tendency to provide functionality that would otherwise have been omitted. Thus, a simple computation will overestimate the savings.

IX. CONCLUSIONS

We expect the use of the Oryx/Pecos operating system to lengthen the useful lifetime of System 75 by permitting easier modification of the applications, portability to other processor families, and expansion of feature content using distributed processing. One advantage of

implementing a PBX on top of an operating system is the potential for future integration with other computing environments; for example, the Oryx/Pecos operating system can provide a *UNIX*™ system execution environment and access to the rich set of *UNIX* system tools and applications.

X. ACKNOWLEDGMENTS

This paper represents the design and development work of many people at the Denver and Holmdel locations of AT&T Information Systems Laboratories. Many of the ideas used in the Oryx/Pecos operating system are adapted from experience with the DEMOS[7] and Thoth[4,14] operating systems and from experience gained from an earlier exploratory project.

REFERENCES

1. C. A. R. Hoare, "Monitors: An Operating System Structuring Concept," CACM, *17,* No. 10 (October 1974), pp. 549–57.
2. R. S. Fabry, "Capability-Based Addressing," CACM, *17,* No. 7 (July 1974), pp. 403–12.
3. B. Liskov et al., "Abstraction Mechanisms in CLU," CACM, *20,* No. 8 (August 1977), pp. 564–76.
4. D. R. Cheriton, *The Thoth System: Multiprocess Structuring and Portability,* New York: North Holland, 1982.
5. E. W. Dijkstra, "Hierarchical Ordering of Sequential Processes," in *Operating System Techniques,* C. A. R. Hoare and R. H. Perott, editors, New York: Academic Press, 1972, esp. pp. 91–3.
6. H. C. Lauer and R. M. Needham, "On the Duality of Operating System Structures," 2nd International Colloquium on Operating Systems, IRIA (October, 1978). Reprinted in ACM Operating Systems Review, *13,* No. 2 (April, 1979), pp. 3–19.
7. F. Baskett, J. H. Howard, and J. T. Montague, "Task Communication in DEMOS," in Proc. Sixth ACM Symp. on Operating System Principles, Purdue University (November 1977), pp. 23–31.
8. T. A. Linden, "Operating System Structures to Support Security and Reliable Software," Computing Surveys, *8,* No. 4 (December 1976), pp. 409–45.
9. T. J. Pederson, J. E. Ritacco, and J. A. Santillo, "System 75: Software Development Tools," AT&T Tech. J., this issue.
10. D. L. Parnas, "On the Criteria Used in Decomposing Systems into Modules," CACM, *15,* No. 12 (December 1972), pp. 1053–8.
11. W. M. Gentleman, "Message Passing Between Sequential Processes: The Reply Primitive and the Administrator Concept," Software—Practice and Experience, *11,* No. 5 (May 1981), pp. 435–66.
12. W. Densmore et al., "System 75: Switch Services Software," AT&T Tech. J., this issue.
13. K. S. Lu, J. D. Price, and T. L. Smith, "System 75: Maintenance Architecture,"AT&T Tech J., this issue.
14. R. Cheriton et al., "Thoth, a Portable Real-Time Operating System," CACM, *22,* No. 2 (February 1979), pp. 105–15.

AUTHORS

Kenneth T. Fong, B.S.E.E., 1970, California Institute of Technology; M.S.E.E., 1971, Stanford University; AT&T Bell Laboratories, 1970–1982; AT&T Information Systems Laboratories, 1983—. At AT&T, Mr. Fong has been involved with exploratory development of business communications systems, development of tools for software development, application of dis-

tributed processing techniques to real-time systems, and development of operating systems software. He is presently Head of the Operating Systems Development department. Member, IEEE.

John A. Melber, B.S.E.E., 1971, M.S.E.E., 1975, Polytechnic Institute of Brooklyn; Naval Air Development Center, 1971–1974; General Instrument Inc., 1974–1976; AT&T Bell Laboratories, 1976–1983; AT&T Information Systems Laboratories, 1983—. At the Naval Air Development Center, Mr. Melber designed automatic test equipment. At General Instrument, he was involved in simulator development, CAD database development, operating system support, and computer center operations. His initial AT&T assignment involved database and operating system work on the ACD-ESS Management Information System (AEMIS) project. In 1980, he moved into operating system work for System 75. He is presently Supervisor of the Common Component Software development group for System 75. Member, ACM, IEEE.

Gary R. Sager, B.S. (Mathematics), 1968, M.S. (Computer Science), 1969, and Ph.D. (Computer Science), 1972, University of Washington; Colorado State University, 1972–1974; Los Alamos Scientific Laboratory, 1978; University of Waterloo, 1974–1979; AT&T Bell Laboratories, 1979–1982; AT&T Information Systems Laboratories, 1983–1984; Sun Microsystems, 1984—. At AT&T Bell Laboratories and AT&T Information Systems Laboratories, Mr. Sager worked on the application of distributed processing techniques to real-time systems and the development of operating systems software.

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

Project Development Environment

By T. S. KENNEDY, D. A. PEZZUTTI, and T. L. WANG*

(Manuscript received July 11, 1984)

The development of the AT&T System 75 office communication system required the coordinated effort of many designers working on a large number of individual components of the product. This article describes the project environment and methods created to accomplish this task. Emphasis is placed on the uncommon aspects of the project: the hierarchy of product specification documents that provided great flexibility in design decisions; the concept of a feature engineer that allowed for the vertical development of a feature by one person from feature specification to software code; the baselining and change control procedures that kept decision making at the lowest possible level; the tracking of progress so that prompt corrective action could be taken as problems arose; and the high reliance on electronic documentation and communication.

I. THE DEVELOPMENT PROCESS

1.1 Overview

The development of the AT&T System 75 office communication system spanned almost three years from product definition to introduction. The process consisted of a sequence of steps including requirements generation, external and internal design specification, imple-

* Authors are employees of AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc.

mentation, integration, system testing, and field validation. The hardware and software components were designed in parallel to shorten the development time. Features were developed in phases so that the software architecture could be stabilized and refined incrementally. This was not unique to the System 75 project—much was learned from the experiences of other AT&T projects, such as System 85, Net 1000, *Dataphone*® II data communications service, *Horizon*® communications system, and the Transport Network, as well as evolving theories of software project management. However, in applying the theory and experience of others, methods were specifically defined to take advantage of a completely paperless office environment, to keep decision making at the lowest level, and to allow design flexibility that would accommodate innovation as the project progressed. This article provides an overview of the development process and emphasizes uncommon aspects that began with the project organization.

1.2 Project organization

The coordination of the effort of many developers working on individual parts of the project contributed to the success of the System 75 project and required as much organizational and managerial innovation as it did technical innovation. A decision was made at the start that the most effective way to organize was to minimize the interorganizational coupling and component deliveries. The functional development organization that evolved had three major *communities*: software, hardware (including both circuit and physical design), and test (including system test and field support). This organizational structure minimized coupling and fostered an *entrepreneurial* atmosphere because it encouraged the ownership of individual component designs and promoted innovation.

Figure 1 shows the simplicity of the component flow. The flow of components followed several major routes. (1) Requirements were generated jointly between the development organization and the systems engineering organization. (2) Hardware and firmware designs, originating from the circuit designers, passed through physical design to the engineering design and information organization, which generated manufacturing information. The models support group used this information to build and deliver circuit pack models to the designers for integration with the software. (3) At the same time, the physical designers transmitted hardware manufacturing information to the factory. (4) The software passed from the software designers through the integration and system test groups, and eventually the factory, undergoing testing at each step. (5) The factory delivered a complete system with both hardware and software for installation at a controlled introduction location. (6) The field support group received new releases

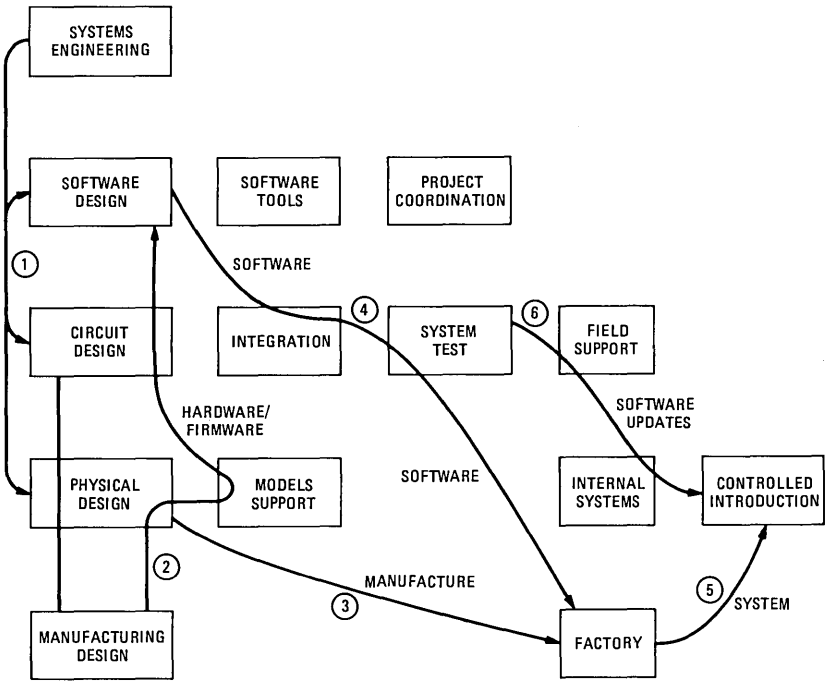


Fig. 1—Primary information transfer.

of software from the system test group, soaked the changes in internal systems, and then installed the new software in the controlled introduction location.

The component deliveries remained primarily within community boundaries. The transfer of integrated components between communities was controlled by *project documents.* The principal documents that were generated are described in more detail later in this section. These documents were generated through a process of negotiation and signature called *baselining*, and changes to the baselined documents (and other components) were closely tracked by a process called *change control.*

1.3 Planning the development

The coordination of planned activities at the project level was the function of the Project Coordination Group. To assure that all major project targets would be met, a *Project Development Plan*, which summarized major project development responsibilities, schedules, resources, and objectives, was negotiated with and signed by the appropriate management.

The major part of the plan was devoted to schedules that listed

important checkpoints or milestones for the project. To simplify the project management function, four schedule levels were defined for milestones on the project:

Level 1—A high-level view of the internal project development containing important checkpoints or milestones.

Level 2—Intermediate milestones, which represented major deliverables from one community to another.

Level 3—Detailed milestones, which represented deliverables between the individual groups within the communities.

Level 4—Internal milestones, which detailed activities of individual developers within the groups.

A computer-based tool called the Milestone Schedule Tracking System (MSTS),* which is described in Section II, was written to maintain and update these schedules.

The Project Development Plan contained Level 1 and Level 2 milestones, and estimates of the total project resource needs. Each community prepared and tracked a *Community Development Plan*, which contained Level 3 and 4 milestones, and detailed community resource estimates. The format and content of the community plans were tailored to meet the specific needs of the community rather than to follow an arbitrary, rigid project standard. By partitioning the schedules in this way, the burden of tracking schedules (i.e., the process of reporting completion of milestones and changes to estimated completion dates) was distributed. Individual communities retained control of their schedules and could adjust them as unplanned events occurred to meet the project commitments contained in the schedule for Levels 1 and 2.

1.4 Hardware development process

The hardware effort involved the design of 18 circuit packs; 2 consoles; and the cabinet, carriers, power, and interconnection arrangements to support these designs. The critical part of the hardware effort involved the design of new circuit packs and their integration with the firmware and software. The hardware community followed a traditional design program of building bread board, brass board, and prototype designs of the circuit packs. Typically, each circuit pack went through two or three design iterations before manufacturing information was transmitted to the factory.

The hardware community development plan contained Level 3 schedules for each circuit pack and hardware component. Critical milestones tracked deliveries to and from the engineering information and design organization for generating circuit pack manufacturing

* Acronyms and abbreviations used in the text are defined at the back of the *Journal*.

information, for availability of first circuit pack models, and for dates of transmittal of the manufacturing design information. The hardware community also held regular teleconference meetings with engineers from the factory to support the introduction of the new hardware designs into manufacture.

1.5 Software development process

A phased development process allowed the software to be built incrementally and avoided the pitfalls of a "big bang" integration. A total of four phases were planned, each lasting about six months. At the end of each phase, the software was delivered to system test and underwent rigorous testing. Critical performance factors and data were measured and the software structure was reviewed by the designers. Each phase was self-contained; as the product was designed and stabilized incrementally, the software architecture was refined.

Software feature completion was scheduled in two parts to avoid freezing the design too early. A Level 3 schedule, which defined the features and other capabilities to be completed during each phase, was baselined at the beginning of the development. A detailed Level 4 schedule for each phase, which defined the capabilities to be delivered per process, was built incrementally as the process design specification of each feature planned in the phase was reviewed and baselined. When the Level 4 schedules were complete midway through the phase, the design of the software components and the implementation effort were well understood.

Software integration was the focal point in scheduling and tracking. Both the Level 3 and Level 4 schedules were kept and tracked by the integration group and the milestones were established based on the integration dates.

1.6 Defining the product

A top-down design and planning process was followed throughout the development of the hardware and software. A comprehensive documentation effort was committed for the software design. The documentation hierarchy, shown in Fig. 2, reflects the two-dimensional aspects of the design process—system feature/capability specifications (external), and system architecture/structure specifications (internal). The activities in brackets <· · ·> represent the relationship of several major activities to the documentation hierarchy.

The *technical proposal*, which defined System 75 at high level and served as the major contract for the development organization, was written jointly by the development and the systems engineering organizations. The systems engineers provided input on feature content and product family planning and used data on market characteristics,

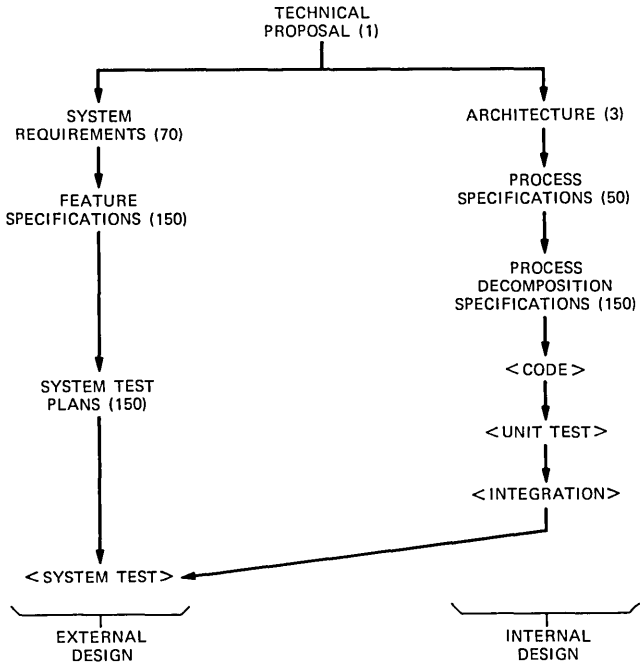


Fig. 2—Software documentation hierarchy.

profitability, and maintainability from the Line Of Business, Business Services, and Marketing organizations. The development engineers provided the technological content.

The organization of the technical proposal was structured around services and features and created the basic product structure for the remainder of the requirements. It specified the feature and system capacity at a high level. For example, it defined what terminals would be supported by the system and gave one-paragraph descriptions of features, such as call pickup, attendant recall, etc. This document was the first to be baselined, and along with the system requirements that followed, it provided the basis for later resource and schedule planning activities.

The next step in the definition of System 75 was generation of *system requirements.* This was a joint undertaking between the systems engineering and development organizations. The requirements expanded the feature definition from the single paragraph contained in the technical proposal to a few pages and captured the essence of the feature. User interfaces and other visible user details were defined with particular attention to achieving commonality with other members of the product family such as AT&T *Dimension*® System 85.

The software designers later prepared detailed external specifications, called *feature specifications*, which were reviewed and approved by the systems engineering organization for agreement with the requirements. The feature specification provided a user-level description of how the feature would be implemented. It included a definition of terms used in the feature operation, a description of feature interactions, and a list of administration requirements.

System test plans were derived from the feature specification and contained detailed scenarios for testing the feature operation.

The *architecture* document, the highest-level internal design document, permitted parallel development by providing conceptual unity in the software design by translating the technical proposal into a functional description of the system. It listed specific processes, libraries, and interface primitives, and defined key relationships between these software modules. It also described common strategies for security, reliability, recovery after failure, performance, and authorizations across the various elements of the system.

The software has a layered structure, with processes as the fundamental building blocks to enforce physical isolation. During the development, a *process engineer* was assigned to each process to review and enforce internal consistency. The *process specification* described the global aspects of a software process and fully specified all externally visible features. It defined how to initialize, invoke, terminate, and communicate with the process. The process specification was written as a set of manual pages. In general, only a few people were allowed to change the files within a process.

Each feature or user service required cooperation of many processes which in turn required the coordination of many designers, each responsible for a particular process. A *feature engineer* broke down each feature into the functionality of the various processes and specified their interfaces in a document called the *process decomposition specification.* It included a definition of how features map into the set of processes or modules, a description of internal process operation, lists of key data stored by the process, and sample sequence of message and process operation to illustrate the feature operation. The design was reviewed by the architecture team for consistency. The detailed integration schedule was committed only after the design was approved by the architecture team.

In summary, the feature engineer was responsible for the vertical design, and the process engineer was responsible for the horizontal implementation. Each was responsible for the software consistency in a particular domain and, together, they expanded the software in parallel. The software development tools[1] supported this process by allowing multiple developers to work in parallel on the software code.

1.7 Integrating hardware, firmware, and software

Several laboratory models of the System 75 were built to support the integration of the hardware, firmware, and software. These models had the major functionality of the final design. A typical model contained a control carrier, a test carrier, interconnection field, power supplies, and terminals. The control carrier had carrier slots on wider spacings to accommodate nonproduction prototype circuit packs, such as wire wrap, and to allow use of adapter and emulators for custom Very Large-Scale Integrated (VLSI) devices that were not yet being manufactured.

The basic skeleton of the models—the processing complex, carriers, power, interconnection hardware, and terminals—was delivered first. Then, the delivery of each new system component, such as a Central Office (CO) trunk circuit pack, was coordinated so that the hardware, firmware, and software elements would be integrated and tested on a single laboratory model before the remaining ones were equipped with the new component. At the start of the project, the delivery of the laboratory models was loosely controlled. As the project progressed, however, the importance of timely delivery of the models became clear and a models support group was formed with the primary assignment to build, deliver, and support the models. This group controlled the models delivery program with a document called the *Models Support Plan*, and used a computer-based inventory database to manage the process.

Figure 3 shows the typical flow of information for a circuit pack from the design phase to its use in the controlled introduction locations. This process was repeated many times as new circuit packs and firmware features were added. The models support group played a key role in assuring that the laboratory models were equipped with compatible versions of the hardware and firmware. The component flows followed the major routes outlined in Fig. 1 and show how a simple management concept becomes complicated when applied to a real problem.

1.8 Delivering the final product

Before the system could be made generally available, its quality had to be assured. It was stressed to the developers that everyone on the project was responsible for quality. However, the burden of proof fell on the system test group.

The goal of the system test group was to find as many faults in the system as possible before controlled introduction began. The system test group based its test plans on the feature specifications generated by the software community. Both manual and automated tests were performed on laboratory models.[2] This testing was supplemented by

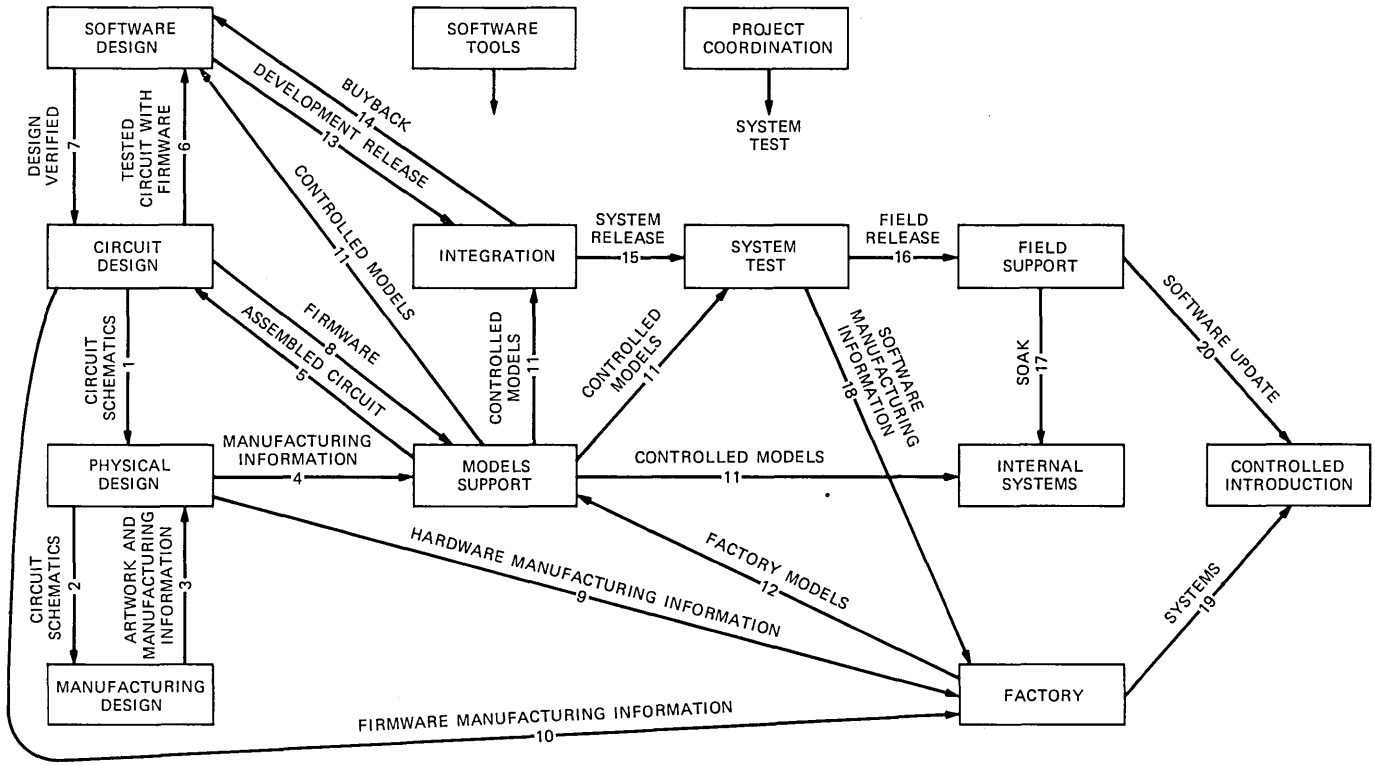


Fig. 3—Information flow for circuit pack design.

providing service to real users on two internal systems. The system test group also periodically released software to the factory. The factory quality assurance organization complemented the system test effort with its own test plans generated from the final product documentation and performed on systems in production at the factory.

The quality was evaluated quantitatively by tracking the number of validated faults against the number of predicted faults. The number of faults was initially predicted using a fault density per lines of software code that was determined empirically from prior developments. This prediction was modified as the actual fault density for System 75 was measured. The number of faults found was validated from data in the Modification Request (MR) system (described in Section II). When the predicted number of faults were found and all service-affecting faults were fixed, the software was ready for release to the controlled introduction locations.

The objective of the controlled introduction was to fine tune the product design and the delivery/support operations before manufacturing large quantities of systems.[3] In addition to evaluating product performance in a field situation, the controlled introduction was needed to test ordering, manufacturing, installation, training, and maintenance processes. Based on the actual field experience, particularly the customer reaction to the product's capabilities and the opinions of sales and service personnel, products were enhanced and/or corrected.

II. THE DEVELOPMENT ENVIRONMENT

2.1 Overview

Office automation based on a *UNIX*™ operating system computing environment was used throughout the project. Development support tools were written not only to help development work, which is commonplace for most projects, but also to improve communications and enforce project methods. Project methods were tailored specifically to be automated by computer-based tools and integrated with the communication services.

A series of regularly scheduled meetings (described in Table I) balanced the electronic communications. The meetings ranged from a semiannual project-wide review attended by all developers on the project to weekly community status meetings attended only by the supervisors.

2.2 Electronic office environment for project communications

The development of System 75 was carried out in a fully automated office environment. Every person on the project from director to clerk

Table I—Regularly scheduled meetings

Meeting	Frequency	Attendees/Purpose
Project review	Semiannual	All Set and maintain a positive mood for the project, make people aware of parts of the project they may have little contact with, and reinforce the team spirit of the project.
Supervisor review	Bimonthly	All supervisors Selected supervisors give a short presentation that focuses on the status of current and near-term deliverables with emphasis on informing the project community of any changes to previously disclosed schedules and of any existing problems that could affect future commitments.
Project status	Monthly	Selected representatives Review schedule commitments, identify problem areas, and make decisions on project-level issues, redistribution of responsibilites, etc.
Software status	Weekly	Software/system test supervisors Discuss and resolve software community issues.
Hardware status	Biweekly	Hardware supervisors/drafting representatives Review drafting status, discuss and resolve hardware community issues.

had a video display terminal, which became as important as the telephone.

The backbone of the office automation service is a collection of services named the Personal Communication Services (PCS). The PCS services are simple enough to be learned by casual computer users yet contain enough functionality to serve the needs of more sophisticated ones. These commands are characterized by a common user interface that is oriented towards a good human interface rather than one easily manipulated by a program. Extensive prompting and feedback are provided, as well as terse forms for more experienced users. Machine-dependent parameters, such as logins, system names, and directories, are hidden. The services can be customized to the personal preferences of each user.

The communication services of PCS fall into several broad categories: electronic mail service, calendar and reminder services, bulletin board service, and other miscellaneous services.

The *electronic mail* service provides for preparing, sending, reading, and filing messages between individuals and groups. Addressing is done by name (e.g., t.j.watson) rather than by the convention of system!login (e.g., hocsh!tjw). A mailing list capability aids individuals in sending information to special interest groups or organizational mailing lists.

On the System 75 project, the electronic mail service was used for a variety of purposes from a simple reminder to the transfer of technical information, either in the form of answers to questions or as a complete

document transfer. The asynchronous aspect of electronic communications eliminated "telephone tag," allowing people with busy schedules to exchange detailed technical correspondence in a timely fashion.

The *calendar* and *reminder* services provide for logging and reminding of future events. A project calendar was created to list project events. This service was also used extensively on an individual basis as a personal time management tool.

The *bulletin board* service provides for posting public messages in a central location. Many special interest bulletin boards were created to list such diverse items as meeting minutes, computer tools news, and want ads. The bulletin board was an easy way to distribute information project-wide and contributed to the feeling of camaraderie that was fostered on the project.

Other miscellaneous services include a profiler command to customize user environment, a directory assistance program, and a message display program that simultaneously displays mail messages, calendar, and a clock in windows on a locked user's terminal.

2.3 Baselining and change control

The Project Coordination Group had the responsibility of keeping the project on track—both in design content and schedule. The philosophy was to do this in an unobtrusive way and provide as much autonomy to the various communities as possible. This was accomplished by establishing formal methods for baselining design information such as requirements and feature specifications, for controlling design changes, and for tracking schedules that balanced the producer's desire for complete freedom of design and flexible schedule dates with the consumer's need for unchanging designs and firm schedule dates.

Baselining is the process whereby the current state of a design is captured to serve as a baseline against which changes can be made. The objective of any baselining process is to ensure that the design has been adequately reviewed to minimize future changes. The baselining process (and change control process) is not unique to the System 75 project; it was adapted from the experience of other projects. The baselining process (Fig. 4) begins with the assignment of a document number, goes through a review process that includes a formal sign-off step, and culminates with delivery of the approved document to the project library.

The review process was the key to the baselining process. Two types of processes were found to be effective: a design review and a circulation review. Both types of review were held at a peer level. The design review consists of a formal meeting that follows a rigid format with roles defined for a *scribe*, *moderator*, *presenter*, and *reviewers*. The

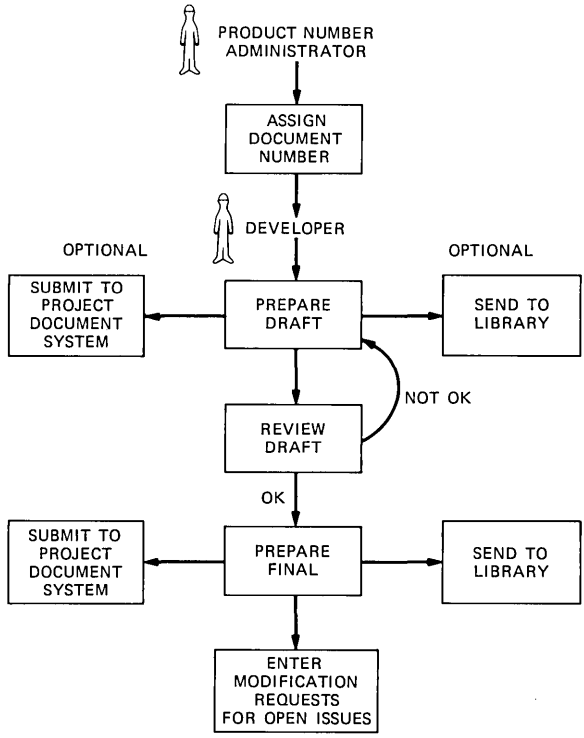


Fig. 4—Baselining process.

circulation review consists of circulating a draft of the document to *primary* and *secondary* reviewers. The primary reviewer consolidates the comments from the secondary reviewer. This reduces the writer's burden of resolving possible conflicts within a community.

The purpose of both types of reviews is to determine whether the document is acceptable (possibly with some changes), unacceptable, or acceptable with some open issues unresolved. In this latter case, the document would be baselined and modification requests entered immediately to record the open issues. This allows a designer to use information in documents that is correct without waiting for all issues to be resolved, thereby hastening the development process.

A project document library was created to serve as a repository for all baselined documents for the System 75 project. Both paper and electronic copies of these documents were kept and were readily available to persons who had a valid need for the information contained in them.

A computer-based Project Document (PD) system was created to retrieve and store copies of the electronic documents and to print a

report of both paper and electronic documents in the library. A database was used to maintain information associated with the project documents and to support the change control process. Both baselined and draft (under review) copies of documents were stored electronically on a single computer system by the PD system. Users on all systems had access to these documents over the computer communications network, providing quick dissemination of information.

Once a document or design was baselined, a formal change control process began. Requests for changes because of either enhancements or errors in the design were tracked via a modification request. An MR is simply a record that contains a description of the resolution of the problem.

The resolution process for MRs (Fig. 5) emphasized the importance of the individual developer in making decisions. In the majority of the cases (the middle route in the figure), MRs were resolved by negotiation between the person assigned and the affected persons—there was no separate review team chartered to approve the resolution of the MRs. The viability of this approach was proven since, during the course of development, developers made few incorrect decisions that required subsequent escalation to correct them.

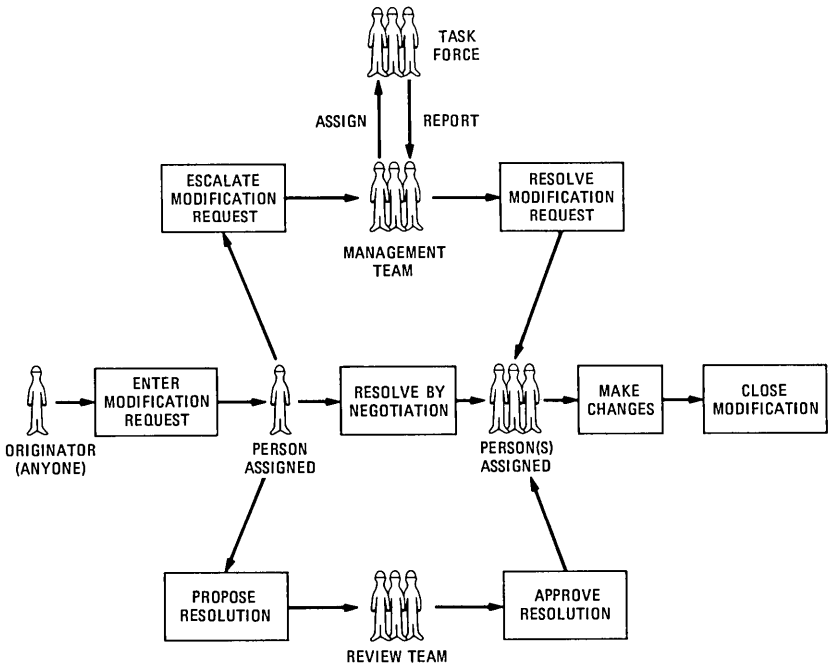


Fig. 5—Resolution process for modification requests.

Some documents, however, such as the technical proposal and system requirements, used the more traditional standing review teams to approve the resolution of MRs. Also, as the product design came closer to completion, additional review teams were formed, especially to assign priorities to fixing bugs and to decide which enhancements should be added. Representation on these boards was limited to a single individual from the relevant communities. Finally, in certain cases, the resolution of MRs was escalated to a management team that would resolve the MRs themselves or create a task force to propose a resolution.

All MRs were processed electronically using the Change Management Tracking System (CMTS) as a tool to store and track MRs. CMTS is a generic system developed by AT&T Bell Laboratories and is currently being used by many organizations throughout AT&T Information Systems Laboratories and AT&T Bell Laboratories. It provides database storage and retrieval of MR information.

To minimize user training, a collection of commands was written to provide a PCS-like interface to the MR database in place of the standard CMTS commands. These commands also allowed greater local control of the MR distribution, required less administrative overhead than the current version of CMTS, operated in a multimachine environment, and provided for completely paperless MR distribution.

2.4 Schedule tracking

Schedule tracking has received a great deal of attention in the project management community, and a large number of computer-based tools, based on an activity network analysis that calculates critical paths, early and late start dates, slack, etc., are available to assist this task. These tools provide valuable information at the start of a project, such as the dependencies of various activities, but can consume large amounts of resources if they are used to track the project as plans change. A simpler approach was undertaken to track checkpoints or milestones. The Milestone Schedule Tracking System (MSTS) was written for this task.

MSTS provides extensive reporting capabilities that have been integrated with the electronic mail service of PCS. The reports can be sorted by date or field data and milestones retrieved based on specific data values. User-defined reports can be created. A typical section of the most common report is shown in Fig. 6.

The milestone schedules were created on a cooperative basis between the various communities. Activity networks, work breakdown structures, and other scheduling techniques were used to create the initial set of consistent milestones. Once the milestones were baselined in

```
Sample                  Milestone Schedule Report                  Issue 2.0
                          Level 1-2 Milestones                      05/03/84

                                                  -----Completion Dates----  s
        Milestone              Cont/Prod/Cons  Original Previous Current  t
        ---------              --------------  -------- -------- -------  -

                            < Hardware Development >

        Interface Circuit Pack
        --------- ------- ----
hw0103  Model Assmbld  TJW/TJW/AGB              04/07/84 04/20/84 --------C
hw0104  Model Tested   AGB/AGB/AGB              04/21/84 05/04/84 05/01/84L
hw0106  EDI            TJW/TJW/dr               04/28/84 05/11/84 05/08/84
hw0107  First Ship     TJW/dr/at&t              12/31/84 -------- --------
                                              >=hw0106 + 8 months: 01/08/85J
```

Fig. 6—Sample Milestone Schedule Tracking System report.

the project development plan, they were updated and reviewed on a monthly basis at the project status meeting.

Each milestone had several key attributes: a *contact*, typically a supervisor, who was responsible for reporting the milestone completion and any changes to the scheduled date; a *description* that briefly defined what completion of the milestone meant (in practice, some incomplete definitions led to disagreement about whether a milestone was complete); and three estimates completion dates—*baselined*, *latest plan*, and *current estimate*. Prior to each review, the contacts provided a current estimated completion date for the milestone. If this date was different than the baselined date, the milestone was discussed at the meeting and, if all agreed, the new date was called the latest planned completion date. If the new date was unacceptable, a solution was devised to complete the milestone on time or to create an alternative plan.

This method, however, proved cumbersome in practice because it was often difficult to agree on schedule changes. MSTS has subsequently been modified to track the *original*, *current*, and *previous* completion date estimates for each milestone. Because this requires less coordination, the tracking interval was reduced from one month to one week to promote faster response to schedule problems. Also, the new method called attention to discrepancies between the current and previous estimates and thereby provided a timely record of project facts.

III. SUMMARY

The development of System 75 drew upon the experience of many other projects that used formal project management methods. Several ideas proved very valuable to the completion of the project: well-defined goals based on a hierarchy of product design specifications and development plans; progress tracked closely so that prompt corrective action could be taken as problems arose; baselining and change

control procedures that stressed keeping decisions at the lowest level possible; computer-based tools specifically tailored to augment the development process; and the degree and timeliness of communications obtained from an efficient, paperless electronic information management and communication service. These ideas are continuing to be used and will be improved for future work on System 75.

REFERENCES

1. T. J. Pedersen, J. E. Ritacco, and J. A. Santillo, "Software Development Tools," AT&T Tech. J., this issue.
2. C. J. Lake, J. J. Shanley, and S. M. Silverstein, "GAMUT: A Message Utility System for Automatic Testing," AT&T Tech. J., this issue.
3. M. A. McFarland and J. A. Miller, "Introduction Activities and Results," AT&T Tech. J., this issue.

AUTHORS

T. Scott Kennedy, B.S. (Mechanical Engineering), 1971, Lehigh University; M.S. (Mechanical Engineering), 1972, University of Michigan; Bell Laboratories, 1972–1983; AT&T Information Systems, 1983—. Mr. Kennedy has worked on the physical design of key telephone systems and the *Horizon*® communications system, and on the development of computer-based tools for project communications. His assignment since 1981 has been as a Member of Technical Staff, System 75 Project Coordination, where he is responsible for planning and tracking and for the development of methods and tools to support these processes.

David A. Pezzutti, M.S. (Electrical Engineering), 1970, Brown University; M.B.A., 1980, Rutgers University; Bell Laboratories, 1969–1983; AT&T Information Systems, 1983—. Mr. Pezzutti has worked on circuit design, software and firmware programming, and in management. He developed and managed the development of a variety of central office maintenance systems for electronic and electromechanical systems that are part of the CAROT System, such as the Remote Trunk Test Unit, Remote Office Test Lines, and Interrogator and Responder technologies. His assignment since 1982 is as Supervisor, System 75 Project Coordination, where he is responsible for planning, tracking, and external product information. Mr. Pezzutti holds five patents. Senior Member, IEEE; member, Sigma Xi, Tau Beta Pi.

Tse-Lin Jack Wang, B.S. (Electrical Engineering), 1964, National Taiwan University, Taiwan; M.S. (Electrical Engineering), 1969; Ph.D., 1970, University of South Carolina; Bell Laboratories, 1970–1982; AT&T Information Systems, 1983—. Mr. Wang was initially engaged in the exploratory work for business communications systems. In 1975, he joined the software development group for the initial *Horizon* communications system development and was appointed Supervisor of that group in 1978. Since 1980 he has worked on System 75 software development. He supervised switch software planning early in the project and led development groups doing call processing and maintenance software and software product delivery. Member, Eta Kappa Nu.

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

Software Development Tools

By T. J. PEDERSEN, J. E. RITACCO, and J. A. SANTILLO*

(Manuscript received July 11, 1984)

Efficient development of high-quality software requires a comprehensive set of software development tools. Tools used within the System 75 office communication system project support a hierarchical model of development, and range from compilers, assemblers, and debuggers to high-level tools that drive the software manufacturing process and aid in monitoring software quality and performance. Tools are applied in each of the many steps required to develop, test, integrate, and maintain product releases. Staff roles and procedures for use of these tools encompass a set of development and release-management cycles that begin with the individual developer and extend into the field support organization. The roles and procedures are flexible and easily customized to support various individual and group assignments. Specific tools to be described include the Object Generation System, the Local Administrative Tool Kit; the Oryx/Pecos Test, Inquiry, and Control System; tools for testing processes in isolation; manufacturing and distribution tools; performance-measurement tools; source-control and change-management tools; and tools for program update and system analysis at field sites.

I. INTRODUCTION

Tools play important parts in every aspect of producing System 75 office communication system software, from initial development and

* Authors are employees of AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc.

unit test to support of systems installed at field sites. A group within the System 75 development organization is responsible for acquiring and developing special tools for the project. Tool developers and product developers are therefore closely allied and draw upon each other's expertise in planning, implementing, adapting, and applying tools as project needs are perceived. This paper presents the results of that collaboration.

Section II provides background on the product hardware, programming languages, and host computing environment from a tools perspective. Section III gives an abstract description of the software development cycle. Section IV describes the software structure that models the System 75 product. The remaining sections describe the subsystem development, project integration, system test and field support stages, along with specific tools and procedures used at each stage.

II. PROJECT ENVIRONMENT

Certain elements of the System 75 project environment have important influence on development support. These elements include processors and operating systems used in the product, programming languages used for product development, computing resources, and size and composition of the software development community.

2.1 Processors and operating systems

System 75 uses 8086, 8088, and 8051 microprocessors from Intel Corporation. The processors serve in different functional roles[1] and vary in size and complexity. For example, the Switch Processing Element (SPE)* is an 8086 with a large main memory, which runs the Oryx/Pecos operating system.[2] The network control and port circuit "angel" processors are 8051's with smaller memories. High-level software functions are implemented in SPE application processes; lower-level functions are implemented in network control and angel processor firmware.

2.2 Languages and compilation tools

The 8086 and 8088 processors are programmed almost entirely in C language.[3] The compiler is supplemented by a large collection of tools that operate on object files. Of particular importance are enhanced linkers that produce multisection object files to model the product operating system run-time environment.

The 8051 processors are programmed in SMAL51, an enhanced

* Acronyms and abbreviations used in the text are defined at the back of the *Journal.*

assembly language. The SMAL (Structured Macro Assembly Language)[4] syntax resembles C in that its instructions are similar to C language assignment statements or "function calls." Higher-level control constructs (e.g., if-then-else, switch) are also provided. SMAL51 thus provides a compromise between the clarity of expression of a higher-level language and the memory space and run-time performance advantages of an assembly language.

2.3 Computing resources

Figure 1 illustrates the computing resources used in System 75 development. Software development and laboratory model support take place on *host computers* that run the *UNIX*™ operating system. Computing resources are divided along organizational, and therefore functional, lines. A high-speed Local Area Network (LAN) permits rapid communication among the host computers, and a shared file system arrangement allows large collections of files to be delivered and shared efficiently.

A test environment consists of System 75 laboratory models supported by the host computers. A laboratory model is an instrumented System 75 with a sufficient variety of terminal equipment to demonstrate feature operation. High-speed data links are used to transfer programs between host computers and models and to access symbol-table information for software testing.

2.4 Software development community

The software community is organized into groups of developers. The number of developers supporting a given processor ranges from only

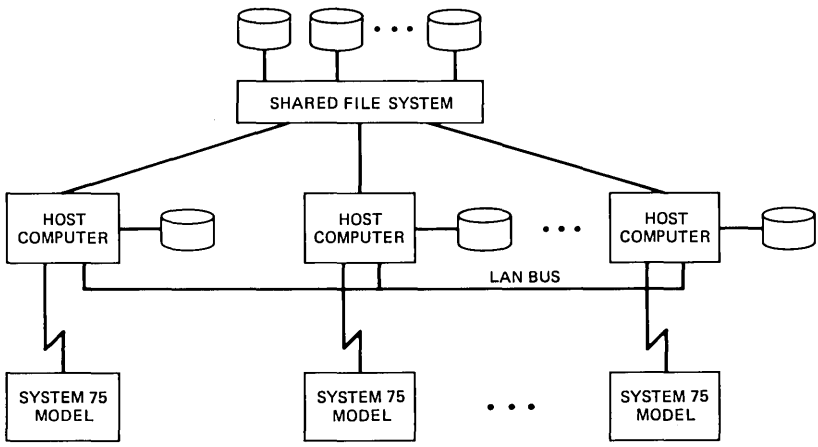


Fig. 1—Computing resources.

a few people for a typical 8051 processor to several groups for the SPE. The SPE groups are organized to correspond to a functional decomposition of the software. Development takes place simultaneously on all components of the system in accordance with a set of product specifications and development plans and a schedule.[5] Milestones in the schedule are planned for collecting, validating, and distributing software among the development groups at key intermediate stages.

Subsequent discussion will center on software development for the SPE. Developers of software for other processors apply a subset of the tools and methods to be discussed.

III. THE DEVELOPMENT CYCLE CONCEPT[6]

The overall process of software production and maintenance encompasses development, integration, system test, and release of a complete product. At a more microscopic level, this process can be viewed conceptually to take place in a set of cyclic activities. Repeated sequences of "develop, integrate, release" steps take place within each cycle. Semantics of the terms Develop, Integrate, and Release differ somewhat among the different types of cycles, and the term "development" is commonly applied to practically any activity connected with software production. However, the following definitions apply generally to the discussion in this section:

- Develop—Create or modify code
- Integrate—Combine and synchronize work of several developers
- Release—Deliver the results to others.

Note that transitions between the steps usually imply satisfaction of acceptance criteria. The development cycle concept is particularly useful because it expresses interfaces among individuals and organizations that deal with the software product.

A somewhat idealized model of System 75 software production is shown in Fig. 2. Stages in the overall process are shown at the left side, and microscopic D-I-R cycles are shown within each stage. A brief explanation follows; later sections explain activities at each stage in more detail. Independent groups of developers build and unit test (D) software components and then deliver (R) the components to project integration. Project integrators combine delivered components from all groups (I), make changes to correct integration flaws (D), and deliver the product back to developers (R). Each development group "buys back" the integrated software into its own environment (I) to test against in producing the next release. Some releases are forwarded from project integration to system test, where the D-I-R cycle models correction of troubles found during final testing (D, I) and delivery to controlled introduction field sites (R).

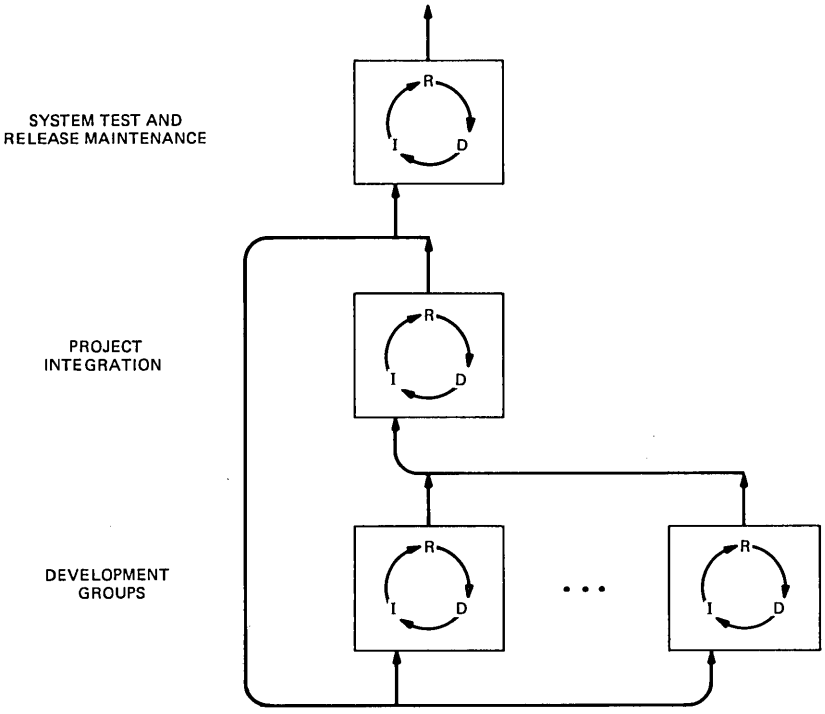


Fig. 2—Development cycle concept.

IV. SOFTWARE STRUCTURE AND TOOLS

A software structure, and tools that build and administer components within the structure, are introduced in this section. An important property of the structure and tools is that they promote, but do not rigidly enforce, uniformity. The project environment, the cyclic model of software development, and the nature of the product itself all influenced the design of these tools. Acceptance of the tools by the software development community was fostered by involving product developers early in the design. Application of the tools at successive stages of software production will be discussed in later sections.

4.1 Object generation system

Because build procedures can become as complex as the product itself, the same build tool should be used at every stage of software production. The primary software build tool used in generating System 75 software is OGS (Object Generation System), an interface to the `make`[7] program supplied with the *UNIX* operating system. The basic components of OGS are conventions for naming files and directories, a collection of makefiles, and a set of build commands.

OGS assumes that source and object code is organized in a uniform directory structure as shown in Fig. 3. The structure has three major levels:

- Book—a directory that contains source and object codes for either a process or a library
- Subsystem—a collection of related books
- Project—a collection of subsystems.

Subdirectories within a book, or subbooks, are also supported. Each directory level and source file type has a unique suffix. Example directory and file suffixes are

`.pj` for project
`.ss` for subsystem
`.p` for process book
`.b` for library book
`.d` for process subbook
`.db` for library subbook
`.o` for object directory
`.c` for C source file
`.h` for header file
`.o` for object file
`.a` for library archive file.

OGS allows code within a single directory structure to be built for

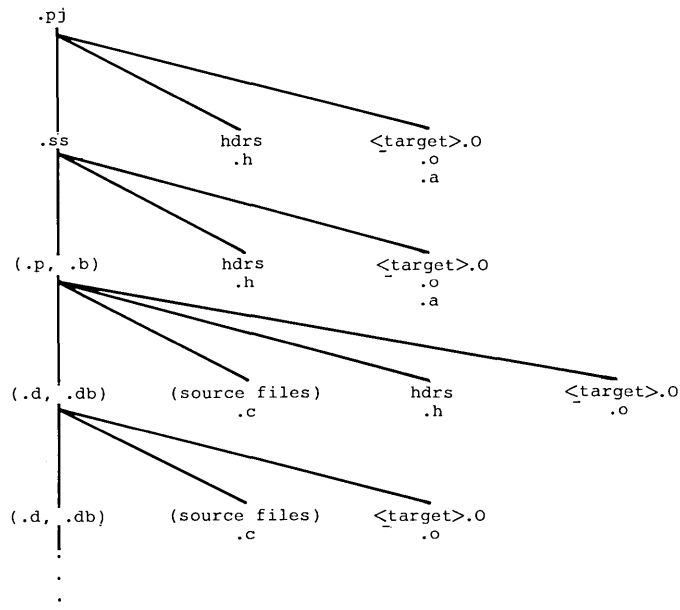


Fig. 3—Generic OGS directory structure.

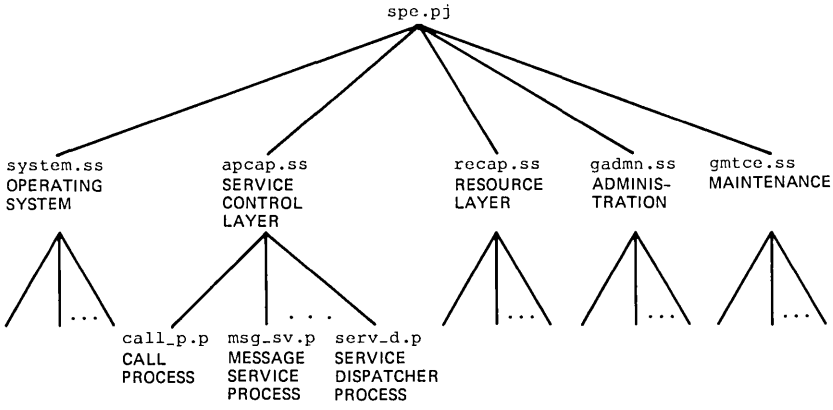


Fig. 4—System 75 OGS directory structure.

several target processors. Object files are placed in target-specific object directories. In addition, the structure incorporates header and library scoping conventions. The scope of a header file is determined by its placement in a book, subsystem, or project "hdrs" directory. Similarly, the scope of a library is determined by the object directory in which the archive is placed.

OGS takes advantage of the uniform directory structure by employing *generic makefiles.* OGS uses one standard makefile per directory level. This has two major advantages. First, all directories at the same level, e.g., all processes, are normally built in the same way. When necessary, build directives can be customized to handle exceptions. Second, developers need not be concerned with constructing makefiles. Once the generic makefile for each level is installed in a standard place, the developer just invokes simple build commands. For example, `ogs mk` (OGS make), executed at the appropriate level, will build a process, a subsystem, or the entire project.

The OGS structure used in developing System 75 software[2,8–10] is given in Fig. 4. Since the directory structure models the software architecture, a developer can easily identify the major architectural components of any subsystem by looking at the directory levels. The scoping conventions for shared headers and libraries facilitate methods for managing changes that may affect several processes. Further, a standard directory structure promotes a generic set of software administrative tools because, like OGS, these tools can infer their operation from the directory level upon which they are executed.

4.2 Local administrative tool kit

LATK (Local Administrative Tool Kit) is a collection of tools that are applied in various activities in the development and integration of

System 75 software. LATK tools operate on OGS directory structures to construct developer work areas, reserve file-edit privileges, submit completed work to an official area, ensure that all subsystems are developed with the same project-level files, verify that OGS structure conventions are followed strictly, and place files under change control.

The functions of the LATK tools in subsystem development, project integration, and system test and field support will be addressed in Sections V, VII, and VIII, respectively. Certain developers are assigned roles that carry responsibilities for coordinating day-to-day work and carrying out routine administrative operations. These roles will be explained in the context of the activities.

V. SUBSYSTEM DEVELOPMENT

System 75 software is logically grouped into the five sybsystems shown earlier in Fig. 4. A subsystem is developed entirely on one host computer. Subsystem isolation is important in a project where a large quantity of new software is being developed. In this environment, the developers of one subsystem are not subject to daily changes in the software of other subsystems. A project integration group periodically brings together and tests the software of all subsystems. Key technical and administrative responsibilities are given to the *subsystem coordinator* and *subsystem administrator* appointed for each subsystem. Project integrators, together with the subsystem coordinators, approve changes to files having project scope, schedule and conduct project integrations, etc.

5.1 Areas

An area is an OGS structure that has been populated with a particular subset of files. *Official subsystem areas* and work areas (or *work spaces*) exist on each host computer. The official subsystem area contains a complete image of the current source and object files for one subsystem and object files for the other subsystems in the project. Changes to this area are controlled by the subsystem coordinator. A work space duplicates a portion of the official subsystem area and contains files that an individual developer is modifying.

5.2 Procedure

A developer sets up a work space for a process book by invoking an LATK tool. The fact that several developers may be working on a given process creates a conflict between the desire to work with the latest copy of each other's code and the need for a stable environment. The work space setup tool copies all of the object code for the process from the official subsystem area, thereby isolating the developer from

changes by others. An LATK tool can be invoked to refresh these object files from the official subsystem area at any time.

The developer uses an LATK tool to restrict edit permission on source files and obtain copies of them in the work space for modification. The OGS build tool compiles these source files and loads them with other object files for the process in the work space. The final load module in the work space can consist of any combination of processes from the work space, the official subsystem area, or even other developers' work spaces.

The developer tests the code in the work space and then uses an LATK tool to submit the work space to the subsystem administrator. The subsystem administrator uses an LATK tool to perform checks on the work space (e.g., file-edit permission checks, directory structure checks, time/date checks on source and object files) and to merge the changed files into the official subsystem area.

VI. UNIT TESTING

Developers must unit test their software before submitting it to an official area. Tools to support testing and debugging are tailored to the product software design. System 75 software consists of the Oryx/Pecos operating system and a collection of application processes that communicate via interprocess messages. Unit test tools for System 75 accommodate testing at both the operating system and application level. At the application level, tools facilitate testing the interactions between processes, as well as testing within a single process.

Three primary test tools are used in System 75 development. They are a low-level monitor for operating system and hardware testing, a high-level symbolic debugger for applications testing, and a process test environment that allows isolated testing of processes in the system.

The monitor and symbolic debugger are part of a laboratory debugging environment pictured in Fig. 5.

6.1 Monitor

The *monitor* is an 8086 debug/test tool that resides in read-only memory and runs as a stand-alone package. The monitor contains mechanisms for transferring 8086 executable files from and to a host computer via either a 9600-b/s asynchronous link or a 50-kb/s synchronous link. It provides a basic software debugging environment with capabilities such as the ability to set and display memory and I/O locations, registers, and memory management descriptors; software instruction execution breakpoints; and single stepping of assembly language execution. As a low-level tool, it supports a simple command language and has no symbolic capability.

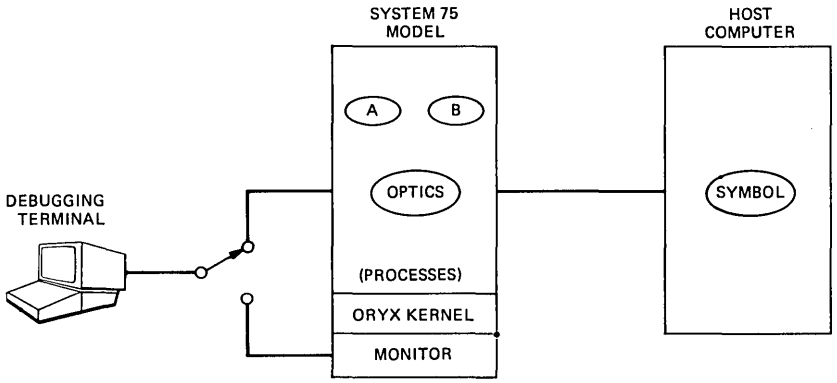


Fig. 5—Debugging environment.

6.2 OPTICS

The Oryx/Pecos Test, Inquiry, and Control System (OPTICS) is a debug/test tool that is part of the Oryx/Pecos operating system. OPTICS has a user-friendly command interface that supports line editing, command completion, and a help facility. It contains a rich set of classical debugging features, such as symbolic referencing, software breakpoints, kernel call tracing, and C stack backtrace. However, the key aspect of OPTICS is that it allows the user to control and monitor Oryx/Pecos processes. Thus, OPTICS can be used not only to debug/test software within a process, but also to debug/test the interactions among a collection of cooperating processes. OPTICS features oriented toward multiple-process debugging include creating, halting, and killing of processes; process status display; and display of a process' path records, path descriptors, and queued messages.

As shown in Fig. 5, OPTICS uses a symbol process on the host computer for symbol-table lookup. The debugging terminal is under control of the OPTICS process when the Oryx/Pecos system is running, and under control of the monitor otherwise.

6.3 Process test environment

The process test environment is a method for testing the internal logic of a process under test in isolation from the rest of the processes in System 75. A sample process test environment is given in Fig. 6. A unit test process and a stub process are available for each real process in the system. The labeled arrows in Fig. 6 represent Oryx/Pecos paths over which interprocess messages are transmitted. In both the unit test and stub processes, program intelligence is replaced by an interactive interface by which a developer can enter data from a control terminal and have it formatted into interprocess messages. A unit test

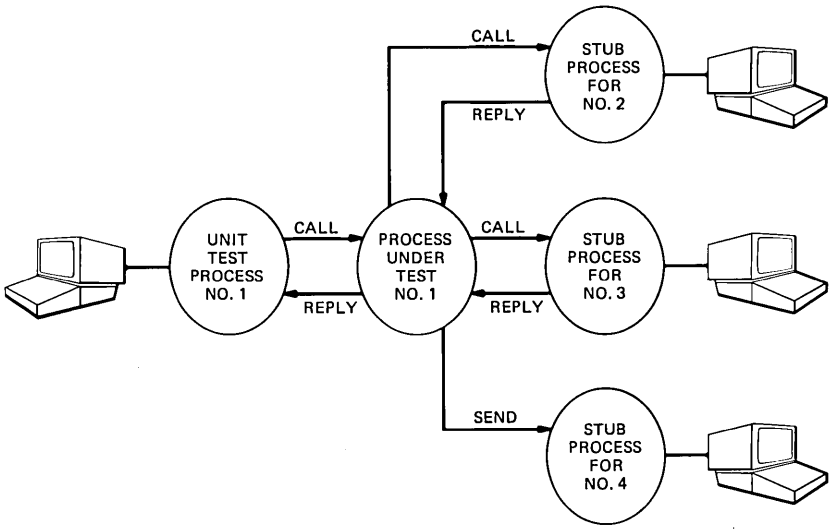


Fig. 6—Process test environment.

process issues the data entered by a developer to the process under test as a message. It can issue all the message types that the process under test can receive and thus can be thought of as a unit test driver. A stub process can be used as a substitute for any process to which the process under test sends messages. It prints at the terminal the contents of a received message, and prompts the developer to enter data to be formatted into a reply message. The functions of the several terminals shown in Fig. 6 are performed by a single physical terminal.

VII. PROJECT INTEGRATION

The System 75 project integration group performs three principal activities. These are combining, manufacturing, and testing software delivered periodically by the development groups (referred to as a *major cycle*); controlling changes to files having project scope (referred to as a *minor cycle*); and other functions such as source code quality checking, performance measurement, and collection of product statistics.

7.1 Major cycle

The major cycle procedure consists of collection from all development groups of formally submitted software, along with specification of the state of development. The main purpose is to ensure that the entire product is consistent, meaning that it can be built with a single set of tools and project-level files and that interfaces between inde-

pendently developed components work correctly. All computable files are rebuilt independently on a separate host computer, and the result is tested and distributed to each development group. The type and degree of testing depend upon the stage of development. In general terms, the aim is to ensure that the quality of the integration delivery is at least sufficient to serve as a base for development of the next stage.

LATK tools support the major cycle by automating routine operations associated with submission and delivery of a large volume of files. An LATK structure verifier tool is used to locate files that do not conform to OGS conventions, and a structure printing tool can be used to get a formatted display of any part of the directory and file structure. Other LATK tools ensure that all submitted software was developed with correct project-level files and report discrepancies. The verification tools are available to both subsystem administrators and project integrators. Serious problems discovered during integration testing are fixed by developers prior to distribution. LATK work space procedures are the same as those followed for subsystem development.

7.2 Minor cycle

Certain software components apply to the entire project and are expressed in files that have project scope. For example, structures of messages used by applications to communicate with the operating system are declared in project-level header files. While it is important to define and freeze files that have project scope as early as possible, it is also important not to delay or hamper development until all such files can be defined completely. Therefore, the minor integration cycle procedure was instituted as a formal means for timely collection and distribution of these files. All minor cycle deliverables are subject to approval by project integration before the distribution takes place. The files are delivered by the development groups and distributed by project integration via shared file systems. The last minor cycle before a major cycle is especially important in that it represents a freeze of all project-level files to be used in the subsequent major cycle.

LATK tools support the minor cycle by assisting subsystem administrators in accepting distributions, validating changes, and ensuring that all components of a distribution are accepted.

7.3 Other functions

As mentioned earlier, project integration conducts some specialized forms of testing. Unlike unit and system tests, these are not functional tests. Instead, they are used to monitor adherence to established criteria for source code quality and performance. The tests are conducted by integrators rather than developers or system testers because

of the need to ensure uniform compliance at frequent intervals. Project integration also serves as a central point for gathering statistics on the product and development process, such as source code and memory usage statistics. Two principal types of specialized testing are source code quality checking and hardware-assisted performance measurement.

7.3.1 Source code quality measurement

A set of `lintogs` tools, based on the `lint`[11] tool supplied with the *UNIX* operating system, are used by project integration to measure source code quality. `Lint` examines C source files and points out syntactic, stylistic, and semantic characteristics that may cause bugs, waste, or portability problems. `Lintogs` has additional capabilities to analyze C source files in an OGS structure and to report "local" characteristics of each file, and "global" characteristics of the collection of files. Local characteristics include such things as conformance to strict type rules and detection of unreachable statements. Global characteristics include inconsistent use of function arguments and return values. In principle, `lintogs` could be applied by every developer. However, the large collection of files makes it more practical to apply the tools centrally and report the results back to developers for examination and possible correction.

7.3.2 Performance measurement

System 75 software is required to meet stringent real-time performance standards. Performance measurements to validate designs are carried out by a hardware/software system known as the *spigot* system.

A diagram of the spigot system is shown in Fig. 7. In brief, it consists of hardware that provides high-resolution timing of a sequence of events reported to it by software under performance test. The software reports an event by executing a "spigot call," which sends information to the hardware. Typical events might represent the beginning and end of handling of a message by a process. An event comprises type and data fields in a spigot data packet. The hardware adds a timestamp field and transmits the packet to a data collection processor, where it is buffered on a disk. The event file is transmitted from the data collection processor to a host computer for analysis at the conclusion of a test run. For example, statistics on each of a sequence of steps required to process an external stimulus might be gathered under varying system loads.

VIII. SYSTEM TEST AND FIELD SUPPORT

Deliveries from project integration to system test occur at major project milestones. Each such delivery represents completion of soft-

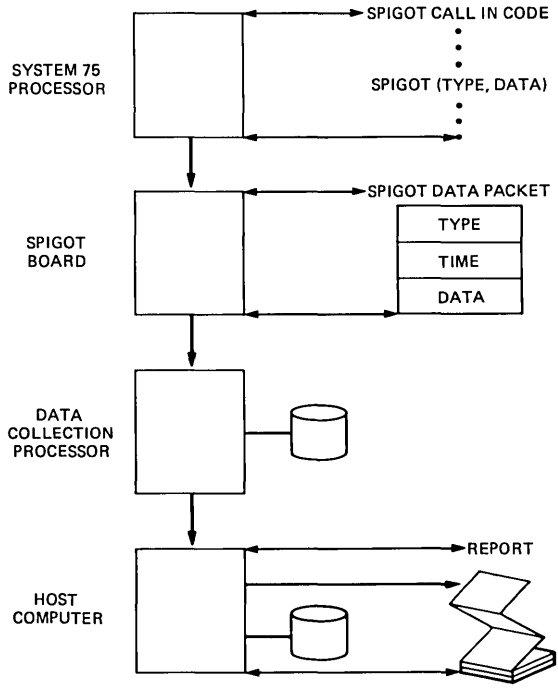


Fig. 7—Spigot system.

ware that implements a well-defined set of product features at a high level of quality. (The principal automated system testing tool, GAMUT, is discussed in Ref. 12.) The first topic of this section is a test coverage system that aids in judging the effectiveness of system testing. The remaining topics deal with tools and procedures to manage and support field releases.

8.1 Software test coverage

System test procedures are designed to exercise a load module as thoroughly as possible to verify correct operation of features. Test coverage is one objective measure of how thoroughly a program under test has been exercised.[13,14] Although more elaborate coverage arrangements exist, a low-cost system that simply records which instructions have been executed is a valuable aid in improving tests.

The coverage arrangement used in testing System 75 comprises hardware and software components. Coverage analyzer hardware monitors the SPE physical address bus. There is one bit of coverage analyzer memory for each byte of SPE memory. During a test run, coverage memory bits corresponding to SPE memory locations that

have been executed are set to '1'. Commands to dump the coverage memory at the conclusion of the test run and to analyze coverage dumps are provided on the host computers. The analysis commands produce a hierarchy of reports that relate the coverage information to the load module under test. A summary report for the entire load module gives percentages of functions and source lines executed for each process. A summary report for a selected process lists which functions and which source lines in each function have been executed. Marked source code and object file disassembly listings can also be produced. A merging program is used to combine coverage data from several test runs and discard obsolete data from parts of the load module that have changed. Of special importance to the system test application is the fact that modification of the load module under test or the conditions under which it runs is not necessary.

8.2 Change control

Development and integration activities occur at a rapid pace, and files are subject to very frequent revision. There has been no perceived need to maintain a history of software changes at these stages of development. However, a system test or field release is a supported product for which strict control over changes is necessary. The two families of tools used for this purpose are the MR (Modification Request) system and the MESA (Management Environment for Software Administration) system.

An MR is a request to modify a product to fix a problem or add a new capability.[5] The MR system is a database for tracking such requests. One use of the MR system is to record problems discovered in system test or field releases. Entries in an MR form contain the originator's analysis of a problem, the severity, the product component, and the release affected. Other entries describe the eventual resolution of the MR. Reports on open MRs are analyzed to judge the importance of fixing each problem and to schedule further investigation and fixes.

MESA is an interface to the Source Code Control System (SCCS),[15] supplied with the *UNIX* operating system. MESA has added capabilities for control of a structured collection of files. That is, in addition to the standard SCCS function of recording changes to individual files, MESA also records information about the version number of every file in a release and its place in a directory structure. Thus any whole release, as well as any version of any file, can be recovered from the "pool" of SCCS files. Tools are provided to handle three types of MESA operations: initial introduction of a complete release of software into MESA, changes to files made in developer work spaces, and definition and reconstruction of releases.

8.3 Release management

By the time a software package has been delivered to the field, several copies of the package exist in different release-management stages. A field copy of the software represents the package installed at customers' sites. A working copy allows the developers to make minor enhancements and fix problems. Two intermediate copies permit system testing and soaking on an in-house system. The software graduates from one of these environments to the next as certain quality levels are achieved. Each of the environments is under MESA control and constitutes a separate view of the same MESA source pool.

In the course of system testing and field experience, MRs are filed against the product. In the normal case, changes are made to the working copy, and SCCS *deltas* are generated. The changes will be delivered to the field when the working copy has become the field copy. When quicker turnaround is desired, a change may be applied to one of the other copies, and a *branch delta* created. To resolve an MR, a developer uses the same LATK, OGS, and unit test tools discussed earlier to create a work space, check out files, build, test, and submit changes. An administrator invokes MESA options in LATK tools to merge files from the submitted work space into the MESA pool.

8.4 Field support

Field support tools are needed for two purposes. One purpose is to administer field software deliveries. The second purpose is to analyze system behavior at controlled introduction field sites[16] when necessary.

Software is delivered to a field site either as a complete reissue of the system tape or as an electronically transmitted update. Tools are provided to generate tapes on a host computer that supports a System 75 tape drive. The tape generation tools are applied by field support personnel during initial trial of a new release and are also used for factory production of tapes. Small updates to a software release can be transmitted electronically. An update is accomplished by a three-step procedure. First, a tool compares an "old" and "new" software release and generates an update file composed of commands to shift and replace software in the System 75 memory. Next, the update file is transmitted to a field system running the old release and is recorded on its tape. Last, commands within the update file are interpreted by System 75 firmware to transform the software into the new release.

Maintenance features within System 75 software are designed to detect and guide repair of system troubles.[10] However, other tools can be applied when further analysis is needed. In particular, the OPTICS tool discussed earlier can be made available at a controlled introduc-

tion site when necessary. Other field support tools provide for off-line analysis of a dump of the processor memory and of translation[9] data stored on the system tape. These tools are especially valuable for studying behavior that occurs only in a particular system configuration.

IX. CONCLUSIONS

The rich set of software tools described here, coupled with extensive computing resources, and a large-scale laboratory models program are all essential to the development of System 75 software. The tools support initial development and unit testing, automate the activities of software manufacturing, and assist in measuring software quality and performance. Methods and tools were defined jointly by the tool builders and members of the software development community. The tools continue to be enhanced when necessary to reflect evolving project needs.

X. ACKNOWLEDGMENTS

This paper presents the work of many individuals at the Denver and Holmdel locations of AT&T Information Systems Laboratories.

REFERENCES

1. L. A. Baxter et al., "System 75: Communications and Control Architecture," AT&T Tech. J., this issue.
2. K. T. Fong, J. A. Melber, and G. R. Sager, "System 75: The Oryx/Pecos Operating System," AT&T Tech. J., this issue.
3. B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Englewood Cliffs, N.J.: Prentice-Hall, 1978.
4. C. Popper, "SMAL—A Structured Macro-Assembly Language for a Microprocessor," *Digest of Papers, COMPCON Fall 74,* 1974, pp. 147–51.
5. T. S. Kennedy, D. A. Pezzutti, and T. L. Wang, "System 75: Project Development Environment," AT&T Tech. J., this issue.
6. D. M. Emerson and G. R. Sager, unpublished work.
7. S. I. Feldman, "Make—A Program for Maintaining Computer Programs," Software—Practice and Experience, *9,* No. 4 (April 1979), pp. 255–65.
8. W. Densmore et al., "System 75: Switch Services Software," AT&T Tech. J., this issue.
9. H. K. Woodland, G. A. Reisner, and A. S. Melamed, "System 75: System Management," AT&T Tech. J., this issue.
10. K. S. Lu, J. D. Price, and T. L. Smith, "System 75: Maintenance Architecture," AT&T Tech. J., this issue.
11. S. C. Johnson, "Lint, a C Program Checker," Computing Science Technical Report #65, Bell Laboratories, Murray Hill, N.J., January 1977.
12. C. J. Lake, J. J. Shanley, and S. M. Silverstein, "System 75: GAMUT: A Message Utility System for Automatic Testing," AT&T Tech. J., this issue.
13. E. F. Miller, Jr., "Program Testing: Art Meets Theory," Computer, *10,* No. 7 (July 1977), pp. 42–51.
14. D. V. Buyansky and J. W. Schatz, "No. 1A ESS Laboratory Support System—Erasable Flag Facility," Proc. 6th Int. Conf. Software Eng., 1982, pp. 279–86.
15. M. J. Rochkind, "The Source Code Control System," IEEE Trans. Software Eng., *SE-1* (December 1975), pp. 365–70.
16. M. A. McFarland and J. A. Miller, "System 75: Introduction Activities and Results," AT&T Tech. J., this issue.

AUTHORS

Thomas J. Pedersen, B.S. (Electrical Engineering), 1960, Iowa State University; M.E.E., 1962, New York University; Bell Laboratories, 1960–1982; AT&T Information Systems Laboratories, 1983—. Mr. Pedersen has worked on a variety of communications research projects and recently on the *Horizon*® communications system and System 75 projects. He is currently working on software tools to support business communication system development. Member, IEEE.

Joseph E. Ritacco, B.S. (Mathematics), 1965, Polytechnic Institute of Brooklyn; M.S. (Mathematics), 1966, University of Michigan; Bell Laboratories, 1965–1977, 1979–1982; American Bell International Inc., 1977–1979; AT&T Information Systems Laboratories, 1983—. Most of Mr. Ritacco's career has been as a system programmer in a computer center environment. His responsibilities included IBM operating system support, performance analysis and tool building, remote job entry, and networking development and operations management. He supervised a group responsible for software tool development for the System 75 project. He is currently supervising a group responsible for developing a software system that performs remote maintenance and administration of AT&T products. Member, ACM.

Jamie A. Santillo, S.B. (Mathematics), 1975, The Massachusetts Institute of Technology; M.S. (Information and Computer Science), 1976, Georgia Institute of Technology; IBM, Poughkeepsie, 1976–1978; Bell Laboratories, 1978–1982; AT&T Information Systems Laboratories, 1983—. Prior to working on System 75, Ms. Santillo did reliability and performance analysis of mainframe computers, and developed operating system and database software. With the System 75 project, she designed development environment and integration tools, and then supervised the development of maintenance software and enhanced switch services software. Ms. Santillo is currently supervising a group responsible for project management, integration, and system test of personal computer/work station software.

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

GAMUT: A Message Utility System for Automatic Testing

By C. J. LAKE, J. J. SHANLEY, and S. M. SILVERSTEIN*

(Manuscript received July 11, 1984)

GAMUT is an automated testing tool that can verify message activity in message-based systems. This tool is unique because it can be customized for application to a variety of systems and used in all test phases of a development cycle. Using a single interface, rather than an array of port-specific interfaces, GAMUT can generate and verify load traffic through scripts, as well as provide an automatic record and playback mechanism for feature testing. This paper discusses the design philosophy and distinctive features of GAMUT as well as its architecture. It illustrates message definition and the script command language by building a sample test script of a station-to-station call on the System 75 office communications system and concludes with a discussion of challenges and problems encountered while using the tool to test System 75.

I. INTRODUCTION

The purpose of system testing is to ensure that a newly developed product performs according to established requirements and satisfies customer needs. One approach to testing is to manually exercise systems according to prescribed test plans; testers augment this approach with tools that automatically apply input and verify operations.

* Authors are employees of AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc.

Such automatic tools frequently replace system peripherals and consequently are application-specific. However, the emphasis on structured design in new products has also changed the perspective of testing and consequently the architecture of the automated testing tools; a single test tool can simulate the activity of all peripherals.[1]

This paper describes an automated testing tool, GAMUT, that monitors and verifies message activity in message-based systems by simulating traffic at message interfaces within the system. GAMUT executes scripts, written in a programming-like language, that represent some feature operation of the system; sequences of stimuli are applied to the system under test and responses are verified automatically by the tool.

This paper discusses the design philosophy and distinctive features of GAMUT as well as the architecture of the tool. It illustrates message definition and the script command language by building a sample test script of a System 75 station-to-station call. Some alternative applications of the test tool are also suggested.

II. BACKGROUND: MESSAGE-BASED SYSTEMS

In the past, switching systems were designed with a well-defined, high-level message interface to peripherals or between processors; for example, in Automatic Call Distribution (ACD)* electronic switching systems there were message interfaces between the main switch and the local peripheral controllers as well as the management-information system.[2,3] However, the use of multiple-microprocessors and multiprocessing operating systems in real-time information processing systems has led to the incorporation of message-based architectures at internal implementation levels. Typically, such systems consist of a network of subsystems, implemented with hardware, software, or combinations of both, which communicate through message interfaces. Requests for service, called stimuli messages, are inserted into a subsystem by users or by other subsystems. The subsystem then produces response messages that may become input to other subsystems or that may be output to users on various system terminals.

In a complex system composed of many subsystems with many terminals, the combinations and sequences of stimuli and response messages are endless; an attractive method to exercise and test the design and implementation of such systems is to have a computer utility apply sequences of input messages to a selected interface and validate the resulting output.

In System 75, in addition to the internal interfaces between software

* Acronyms and abbreviations used in the text are defined at the back of the *Journal.*

subsystems, there is a complex interface between the system and communication terminals connected to it. Prior to connecting actual users to the system, it is necessary to test overall system performance under a wide variety of conditions and loads. Problems arise when these tests must be performed prior to the availability of either the interface circuits or the actual terminals.

Other problems, such as simulating actual communication connections or load conditions, occur when the system to be tested is complete but before it is practical to actually place heavy call volumes through the system. Another case is finding transient faults or faults that result from long sequences of state transitions.

Regardless of the application, systems designed with a message-based architecture contain independent subsystems and message interfaces that have the following characteristics:

- Control and data are passed in the form of messages.
- Activity at the message interface defines the operation of the subsystem.
- No subsystem possesses knowledge of how other subsystems perform their functions.
- Since the only way subsystems interact is through messages, the message interfaces form protective fire walls.
- By examining a trace of stimuli and response messages, one can determine if the system is operating correctly.

GAMUT exploits these characteristics and thus provides a powerful test tool for performing feature, load, regression, and range testing.

III. OVERVIEW—GAMUT FEATURES

GAMUT is a message utility that exercises and tests message-based systems by replacing part of the system at an interface boundary or by operating in parallel with the system. It interfaces with such systems either by processing messages entered interactively or by executing test scripts. A script is a file that contains sequences of stimuli to insert into the system under test, response messages to verify, and control information. The content of a script, taken as a whole, represents some feature operation or system function.

Since GAMUT is based on an architectural property of a class of systems and not on the design of a particular system, it differs in many ways from other test utilities.[4,5] It can keep pace with a product's development and provide unit, integration, system, and field-test functions.[6] The tool can be customized for application to a variety of systems by selecting an appropriate access interface and by creating a message database. It accepts parameterized scripts, by using scalar, array, and don't-care variables, for repeated applications of test cases,

and can generate traffic load tests using a single interface rather than an array of port-specific interfaces.

GAMUT consists of two major components, common message-utility software and a message-access module. The first part, the larger of the two, generates, validates, and manipulates messages and essentially remains unchanged for any test application. On the other hand, the access module connects the common software to the system under test—inserting and recording messages passing through a message interface of the system under test—and is usually specially designed for each test application. Thus, GAMUT can be applied to different test applications by customizing the message-utility software and "plugging-in" the appropriate access module. Other test systems can be too closely married to the system under test to permit this.

Since the utility can be configured to simulate a subsystem, it can be used early in development as a unit test tool. In addition, the subsystem simulation capability allows GAMUT to be used as a temporary replacement for unavailable subsystems or terminals. As development proceeds and more subsystems become available, the tool can remain in place and be used as an integration test tool. When the development reaches its final stages, GAMUT can be moved to the major message thoroughfare and drive the design through system tests. This is impossible to achieve with a tool that is coupled to a particular system point, and development teams usually have to develop different tools for each phase of testing.

When GAMUT is connected to a major message interface, it can be used to insert multiple, near-simultaneous stimuli and verify system performance. Other test systems can be used for load testing as well, but usually these tools simulate actual user activity and require a large number of interfaces to achieve this and are, therefore, as complex as the system under test. This tool achieves load testing more economically by driving one key interface internal to the system under test. Furthermore, since it can exercise independent subsystems, GAMUT can apply subsystem load tests to identify bottlenecks that limit system performance.

GAMUT can be configured to operate in parallel with the system under test. With this arrangement, it can be used to monitor activity on an operational system for troubleshooting or for acquiring measurements. This arrangement also allows testers to inject both simulated and manual traffic on a laboratory system to study feature interactions and load performance.

The ability to record activity, store the messages, and play them back at a later time provides an additional benefit for system testers. Since systems are usually developed in phases for field release, both old and new features must be tested in each phase. GAMUT is used

to record manual test cases for automatic retest of features at a later date, thereby guaranteeing continued system quality.

Thus, by taking advantage of the properties of message-based systems, GAMUT provides a foundation for multiple testing perspectives, provides testers with leverage in applying a wide variety of tests, and encourages a phased, systematic testing philosophy consistent with the architecture of the system under test.

In addition, because of the tool's structure, its application is not limited to testing systems designed with message-based architectures. By developing an access module that taps onto a collection of key system signals—including perhaps a clock signal—a message interface can be presented to the message utility system where there was no interface before. Test scripts can be devised and applied to the access module, which in turn maps the messages to sequences of signals at the appropriate connection.

IV. GAMUT ARCHITECTURE

A block diagram of the GAMUT test system is shown in Fig. 1. Within a system under test, a key message interface between subsystems A and B is selected and made available to a GAMUT access module. GAMUT will apply test stimuli and monitor system responses at this point to determine if the system is performing as expected. Typically, in System 75, subsystem A was the central control complex and subsystem B was the switching network and port complex. The message interface between A and B was the control channel.[7]

4.1 GAMUT access module

The access module interfaces the system under test to the message utility portion of GAMUT. Depending on the application, the module can be as simple as an Electronic Industries Association (EIA) channel. However, usually a special-purpose, microprocessor-based access module with interface-specific circuits is required. The access module provides five major functions:

1. Inserting stimuli messages into a subsystem under control of the message-utility system.
2. Capturing response messages and transmitting them to the message-utility software.
3. Message time stamping.
4. Peak traffic buffering.
5. Real-time message filtering so that only requested messages are recorded.

It is important that the module operate in real-time and log messages without affecting the performance of either subsystem, as well as to insert messages indistinguishable from subsystem-generated messages.

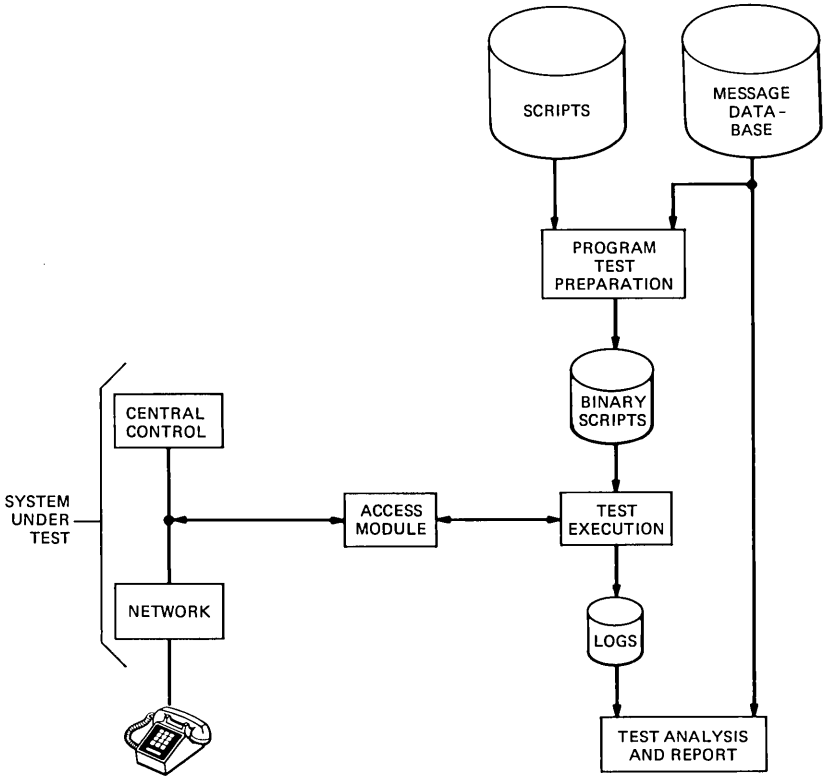


Fig. 1—GAMUT test system.

4.2 Common message-utility software

The common message-utility software runs under the *UNIX*™ operating system. It is driven by a database that contains a template for every message that flows between subsystems A and B. Isolating this format information in a database allows GAMUT to track message changes or additions within the interface and facilitates moving to another interface. This database allows users to communicate with the message utility software in a symbolic message language; it allows the software to convert this language into the format of the system under test; and it allows the software to report test results in the original format. Therefore, before any testing can begin, a description of every possible message that will flow through the message interface must be entered in the message database.

Test cases are entered into scripts that contain sequences of stimuli messages, expected system-response messages, and control information to specify how the messages are to be applied. Scripts are compiled by test-preparation software that converts the messages in the script

language into the format expected by the system under test. Control information is converted into a form easily interpreted by the test-execution software. The resultant file is called a binary script.

Binary scripts are then passed to the test-execution module, which can execute more than one binary script at a time to allow simulation of realistic test scenarios. The execution module reads the binary scripts and interprets script commands much like a computer CPU executes instructions; it cycles through multiple scripts in a round robin schedule much like a time sharing operating system. When the execution module encounters a stimuli message in a script, it instructs the access module to insert this message into the system under test. This module also monitors messages that return from the access module and verifies that these are valid responses as specified in the script or are exceptional conditions that were not specified. In either case, all stimuli and response messages are saved in log files as the test execution progresses.

When test execution is completed, the tester uses test analysis software to examine the logs. The analysis software consults the message database to convert the raw messages back into the form used by the test designer in the script language. Three types of reports are available. The first is a trace of all messages inserted during the test. The second is a trace of all the inserted messages and responses that provide a step-by-step record of every transaction occurring during the test. The third report is an exception report, which contains messages returned from the system under test that were not specified in a script as valid responses, as well as all messages that were specified in the script but were not received. The last report is interspersed with any diagnostics generated by the script execution. By reviewing these reports, the tester can determine how the test progressed, if the system performed correctly, and if not, what went wrong. A log-filter program is also available to help the tester focus on a particular type of message or portion of the log.

V. MESSAGE DEFINITION

GAMUT is customized to a particular application by defining a message database, called the ASCII Message Definition Table (AMDT). The AMDT is an ASCII file that defines the I/O interface language for the system under test; i.e., the file contains field descriptions of every message that passes through the message interface.

The AMDT contains both general message identification (e.g., op-code, message length, number of fields) and specific field-description information (e.g., field name, default values, and conversion operations). Data are entered in the file with any *UNIX* system text editor according to a prescribed format. Once a template is defined for each

possible message in the system under test, a tester can design tests that combine the messages with GAMUT script commands.

In the next few sections, we consider a script that tests a station-to-station call on System 75. In System 75, circuit packs communicate via messages with call-processing software over a time-division-multiplexed bus.[8] For example, when a user pushes a button on a telephone set, a message is sent to the call-processing software, which generates such responses as application of dial tone and activation of a call-status lamp. The messages are defined by the switch architecture; the off-hook message might be defined as follows:

```
offhook  station_id
```

A typical use of this message might be

```
offhook    1
```

which represents "go off-hook on station number 1." When converted according to the template in the message database, the resultant binary message is ready for insertion in the system under test. Conversely, a binary message received by the test-execution software is converted back to a "script language" message by comparing the messages in the database for field matches and selecting the best match.

VI. SCRIPT LANGUAGE

GAMUT allows a test designer to generate messages, apply them as stimuli, validate the resulting response messages, as well as control test execution. To achieve this, the script language must combine aspects of a traditional programming language, like C,[9] with a language focused on message manipulation. To simulate a particular feature operation, the tester combines a number of scripts into an experiment that GAMUT will execute.

In a script, stimuli messages are combined with the `send` command; the test execution software will insert each `send` message into the system under test. Expected response messages are combined with the `wait` command and for each `wait`, the test execution software will verify that the specified message actually occurred. There are several variations of the `wait` command that provide capabilities for error detection, for verification of a set of messages independent of their order of arrival, and for specification of several possible responses, only one of which will be matched. The GAMUT script language also includes several commands that provide flow control, assignment and arithmetic or logical operations, and subroutine capability similar to that of a programming language. Thus, a test designer can use the script language to specify a feature operation and validate that operation in a variety of scenarios.

6.1 Script execution

An arbitrary collection of scripts can be run together to simulate a field environment and create a realistic test. When these scripts are executed, the operations are verified independently; this can also occur while the system is manually generating traffic. Random characteristics can be added by using a pseudo-random number generator programmed into the script. Frequently one script will be assigned a control function, initializing and dispatching foreground and background tests, waiting for test case completion, and summarizing results. The tests can be executed sequentially or simultaneously. Another script can be assigned an administrative function, varying system translations[10] and corresponding GAMUT variables so that tests can be reapplied over various translation ranges and combinations.

6.2 Sample script

To describe features of the script language, we build below a sample script that checks the ability to place a call from one station to another within System 75. The station numbers and the call duration are established external to the script. The script defines the features of the script command language as necessary; so that the script can be reapplied to a variety of system states, field values—which may vary from one execution to another—will be parameterized.

All variables must be declared before use and can be either scalars or arrays. GAMUT uses the macro capability of C; therefore, `#include` statements can be used to access standard header files and `#define` statements provide simple or parameterized text replacement. Once the variables have been declared, most scripts require some start-up procedure—either to synchronize execution with other scripts in the experiment or to synchronize message flow with the application. Scripts are executed in a prespecified order according to a round robin schedule. For this example, we assume that a control script is executing and that it will initiate execution of the call script by setting the `Run_mode` variable to `RUN`, after defining the originating and terminating stations as well as the call holding time.

Thus the first lines of the script include variable declarations and run-time synchronization:

```
#include   "field_codes"
#include   "error_codes"
#define    RUN 1              # Run-time synchronization
#define    CALL_COMPLETE 2    # Run-time synchronization
#
global     Run_mode           # Script execution state
global     Orig               # Originating station id
```

```
global    Dest              # Destination station id
global    Extension[4]      # Digits of extension array
global    Wi                # Wait timeout interval
global    Holdtm            # Call duration
global    Good_calls        # Number of successful calls
global    Bad_calls         # Number of call failures
global    Busy_calls        # Number of busy calls
#

# Idle while Run_mode is not equal to RUN
#
:idle    if Run_mode != RUN    :idle
```

The values of the global variables are assumed to be set in the control script and, since they are global, will be known to all scripts in the experiment. The line with the `if` statement tests the value of `Run_mode`, and as long as that variable is not equal to the value `RUN`, control remains at the line labeled `:idle`. Once `Run_mode` equals `RUN`, control transfers to the next statement.

Next the station identified by the value of `Orig` will originate a call to the destination station. A message will be sent by the simulator to the call-processing software indicating that the `originator` station went into an off-hook state. The script expects, in response, a message for dial tone to be applied to the originating station. If the response does not occur within a specified time, control will be transferred to the statement `:no_dialtone`, where an error subroutine will be called. The subroutine will be included at the end of the script and will output an error message, increment an error count, and terminate the call.

```
  send                    offhook  Orig
  wait  Wi  :no_dialtone  tone     Orig  DIALTONE_ON
                          .
                          .
                          .


:no_dialtone
  call  :error (NO_DIALTONE)
  goto  :reset
```

On the `wait` command line, the variable following the `wait` is the amount of time (in seconds) the script will wait for a message from the application before logging a time-out error. As soon as the expected message arrives, the simulator processes the next instruction so the wait time should reflect the maximum expected delay between messages. If a time-out occurs, the simulator transfers control to the

statement with the label indicated. To make the script more general, we will assume the wait interval `Wi` is initialized in the control script.

The originator can now dial the destination extension; the digits to be dialed are stored in the array `Extension`. After the first digit is dialed, dial tone should be removed, and the script verifies call processing's action with a `wait` command.

```
  send                dial  Orig  Extension[0]
  wait  Wi  :no_quiet tone  Orig  DIALTONE_OFF
  send                dial  Orig  Extension[1]
  send                dial  Orig  Extension[2]
                      .
                      .
                      .

:no_quiet
  call  :error (NO_QUIET)
  goto  :reset
```

The three digits of the destination extension have now been dialed. If a dialing error occurs, control will be transferred to the statement labeled `:no_quiet`. Note that the same `tone` message is expected when origination occurs and after dialing the first digit; only the field value changes to control the tone.

Two different responses are now possible: the destination alerts or the originator hears busy tone. The `swait` allows the script writer to specify a set of alternative responses, only one of which will be correct. When that response is selected, control is transferred to the statement indicated. If no response is matched, control is transferred to the statement indicated on the `swait` line `:no_call`.

```
  swait  Wi  :no_call
  <
    :hangup  tone  Orig  BUSY_ON
    :alert   tone  Orig  RINGBACK_ON
:alert
  wait  Wi  :no_ringing  ringing  Dest  RINGER_ON
                         .
                         .
                         .

:no_call
  call  :error (NO_CALL)
  goto  :reset
:no_ringing
  call  :error (NO_RINGING)
  goto  :reset
```

A busy response will cause the script to transfer to statement `:hangup`; otherwise, the originator will hear ringback and the script will branch to `:alert` where the application of ringing at the destination is verified.

When the call is answered, several responses (ringback off, ringing off, connect) should occur and be verified; however, the order in which they occur is not important. The `mwait`, multiple message wait, allows the tester to specify all those responses. If at least one of the specified messages does not occur, control will be transferred to the statement specified on the `mwait` line.

```
:answer
  send            offhook     Dest
  mwait  Wi      :bad_ans
  <               tone        Orig       RINGBACK_OFF
                  connect     Orig       Dest
                  ringing     Dest       RINGER_OFF

  >
  delay  Holdtm
                              .
                              .
                              .
:bad_ans
  call  :error (BAD_ANSWER)
  goto  :reset
```

When the destination answers, ringback and ringing are discontinued and the two stations are connected. The connection will be maintained for `Holdtm` seconds, at which time the call will be torn down.

```
  send                        onhook      Dest
  wait   Wi    :bad_onhk      discon      Dest
:hangup
  send                        onhook      Orig
  swait  Wi    :bad_onhk
  <
               :g_done        discon      Orig
               :b_done        tone        Orig  BUSY_OFF
  >
:g_done
  Good_calls = Good_calls + 1
  goto  :reset
:b_done
  Busy_calls = Busy_calls + 1
:reset
```

```
  Run_mode = CALL_COMPLETE
  goto  :idle
:bad_onhk
  call  :error (BAD_DISCONNECT)
  goto  :reset
endscript
```

The destination hangs up first and then the originator. The originator hang-up is labeled by `:hangup` so it can be referenced whenever a call attempt results in busy, as well as for normal call termination. At the completion of the call, the `Run_mode` is reset, and the script will idle again until reinitiated by the control script.

The error subroutine can be added to the end of the script file or included as a separate file.

```
# Error Subroutine
:error  sub (error_code)
  switch (error_code)
    {
    case DIALTONE:
      pstring "Call origination failure"
      break
    .
    .
    .
    case NO_RINGING:
      pstring "Destination alerting failure"
      break
    case BAD_ANSWER:
      pstring "Answer failure"
      send  onhook Dest
      break
    }
:terminate
    send  onhook Orig
    Bad_calls = Bad_calls + 1
  endsub
```

For each specified error code an appropriate error message is output and the originator hangs up. In the case of an answering failure, the destination station is also hung up.

VII. USING GAMUT WITH SYSTEM 75

GAMUT was developed in parallel with System 75 by the Product Test group of AT&T Information Systems. Extra effort was expended

so that operational milestones of the tool and the product were synchronized to allow early testing. This paid off by helping the test group to find protocol, feature, and performance affecting bugs very early in System 75's design cycle. In addition, developers used the tool to diagnose and isolate problems on early models of the switch.

One problem the Product Test group had with the tool was the oversensitivity of the test scripts to message ordering. The early version of the GAMUT script language had no `mwait` capability, and the test scripts explicitly verified message ordering by sequences of `wait` commands. Very often this ordering is not important (e.g., the ringer off, ringback off, and connect sequence from the sample script): as System 75's software evolved, these sequences changed and regression scripts would fail needlessly. Therefore, the product test group added the `mwait` feature to keep GAMUT in step with the product.

Early test scripts were developed by test engineers who had a detailed understanding of System 75's architecture, particularly the control-channel messages. However, as the feature list grew, the number of scripts required to test the system grew. Soon there were not enough skilled testers to develop the scripts. To solve this problem, we built a number of software tools into the utility that provided a record and playback capability that allowed less experienced testers to automate regression test cases in a timely fashion. A feature tester could execute a test that operates correctly on the system peripherals and record the control-channel messages. Next a script generator would read the logged messages and output a parameterized script. When executed, the script would reverify that test case automatically on subsequent releases of the system, as well as provide a basis for a class of related tests (range, load, etc.).

Sometimes, as features were finalized, there were sufficient design changes to invalidate previously recorded scripts. To avoid manually rerecording a test case, we developed a tool that extracted stimuli from the original script and automatically created a regeneration script. This script, with no verification capability, is executed and a revised regression script can be generated with the same software described above.

To test the fault detection and recovery capability of System 75,[11] we replaced the standard GAMUT access module with a hardware fault inserter. Then we wrote scripts to inject faults, execute system maintenance commands, and log test results, thereby extending the tool to System 75 maintenance.

GAMUT's System 75 access module is totally passive in the recording mode and is very portable. We were able to move some of the low-level message handling software from a minicomputer running the *UNIX* operating system to a portable PC, which resulted in a powerful

field tool. Our field-support team has found the portable version of GAMUT to be one of their main tools. It has helped them trace and isolate problems during our controlled introduction, as well as to measure traffic and customer use of features.

We found testing with GAMUT to be automatic, repeatable, and easily documented because test scripts, exception logs, and result logs are automatically saved in *UNIX* system files in the tester's language. Use of this tool facilitated problem isolation and encouraged phased, systematic testing, thus improving the testing process and system quality.

REFERENCES

1. G. J. Myers, *Software Reliability, Principles and Practices*, New York: Wiley, 1976.
2. H. A. Lanty, D. J. Morgan, and H. Oehring, "No. 1 ESS Furnishes ACD Service," Bell Lab. Rec. (March 1978), pp. 76–82.
3. R. J. Jakubek and S. M. Silverstein, "Microprocessor Control of Customer Premises Telecommunications Equipment," Proc. Ann. Conf. ACM (October 1976), pp. 270–4.
4. T. A. Dolotta et al., "The LEAP Load and Test Driver," Proc. Second. Int. Conf. on Software Eng., 13–15 October 1976, pp. 182–7.
5. D. A. Bennett and J. D. Walker, "System Testing the Small Communications System," unpublished work.
6. G. J. Meyers, *The Art of Software Testing*, New York: Wiley, 1979.
7. L. A. Baxter et al., "System 75: Communications and Control Architecture," AT&T Tech. J., this issue.
8. W. Densmore et al., "System 75: Switch Services Software," AT&T Tech. J., this issue.
9. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Englewood Cliffs, NJ: Prentice-Hall, 1978.
10. H. K. Woodland, G. A. Reisner, and A. S. Melamed, "System 75: System Management," AT&T Tech. J., this issue.
11. K. S. Lu, J. D. Price, and T. L. Smith, "System 75: Maintenance Architecture," AT&T Tech. J., this issue.

AUTHORS

Carole J. Lake, B.A. (Mathematics), 1968, Gettysburg College; M.S. (Numerical Science), The Evening College of The Johns Hopkins University, 1970; National Security Agency, 1968–1974; Bell Laboratories, 1979–1982; AT&T Information Systems Laboratories, 1983—. Ms. Lake was initially involved in network operations support system development and for the past four years has worked in system testing for System 75. Member, ACM, IEEE.

James J. Shanley, B.E.E., 1961, Polytechnic Institute of New York; M.E.E., 1963, New York University; Bell Laboratories, 1955–1982; AT&T Information Systems Laboratories, 1983—. Mr. Shanley's development experience includes both hardware and software design on a variety of projects including T1 carrier, Subscriber Loop Multiplexer, *Horizon*® CMS, E911, *Dataphone*® II, and System 75. His current assignment is System 75 Field Support Group Supervisor. Member, Eta Kappa Nu, Tau Beta Pi.

Steven M. Silverstein, B.S. (Electrical Engineering), 1973, M.S. (Electrical Engineering/Computer Science), 1974, Polytechnic Institute of Brooklyn; Bell

Laboratories, 1973–1982; AT&T Information Systems Laboratories, 1983—. Mr. Silverstein has been engaged in business communication and management information system development. More recently he has been involved in system testing for System 75, and he currently supervises the Product Test and Evaluation group. Member, IEEE, ACM, Tau Beta Pi, Eta Kappa Nu.

AT&T Technical Journal
Vol. 64, No. 1, January 1985
Printed in U.S.A.

System 75:

Introduction Activities and Results

By M. A. McFARLAND and J. A. MILLER*

(Manuscript received July 11, 1984)

In today's competitive world, new and complex communications products need to move from the laboratory development environment to the marketplace in a quick but orderly fashion. The objective of a controlled introduction is to evaluate not only the specific product's performance but also its documentation, training, manufacturing, delivery, service, and customer satisfaction. This paper describes the scope of the controlled introduction of the System 75 office communication system. Topics include customer selection criteria, sales-team support, customer and service training, initial customer and internal corporate installations, and the evaluation process.

I. INTRODUCTION

System 75 is a complex and sophisticated business communications system comprised of advanced software, firmware, and circuits.[1] To ensure that its design met customer needs and was reliable, an orderly but quick-paced introduction program was essential. To provide structure and definition to all the activities of this testing program, a System 75 controlled introduction plan was developed.

This paper reviews several of the major activities associated with the introduction of System 75 and specific results that enhanced the

* Authors are employees of AT&T Information Systems Laboratories, an entity of AT&T Information Systems, Inc.

product and improved the effectiveness of its delivery and support methods. Section II describes the overall objective of the controlled introduction. Section III focuses on both customer and AT&T support team training. Section IV presents findings from the early installations. Section V describes items related to maintainability. And, finally, Section VI highlights customer and sales team feedback on the system's capabilities and features.

II. CONTROLLED INTRODUCTION OBJECTIVE

The objective on the controlled introduction was to fine tune both the product design and the delivery/support operations before manufacturing large quantities of systems. Prior to early 1984, System 75 had been tested extensively in the laboratory but had not yet been in use under true marketplace conditions. The controlled introduction, in addition to evaluating product performance, was needed to test the ordering, manufacturing, installation, training, and maintenance processes. Customer reaction to the product's capabilities was also important, as were the opinions of the AT&T sales and service personnel. Based on actual field experience, enhancements and/or corrections were added to the product or methods as necessary.

2.1 Principles of controlled introduction

The System 75 controlled introduction's duration and scope was based on experience from previous PBX introductions.[2] This experience suggested that approximately 60 system-months of in-service testing after system testing were needed to uncover any remaining software or hardware defects affecting production. To meet this criteria, a program was developed that included:

1. Twelve customer installations and two internal AT&T systems, one at Lincroft, New Jersey and the other at Holmdel, New Jersey. The actual deployment is shown in Fig. 1. While a large number of system-months was desirable, there was also a need to limit the number and location of installed systems because of factory production start-up constraints, inventory risk management, a limited number of trained support personnel, and a desire to have quick response time to correct field problems.
2. A stringent set of customer selection criteria (Table I). Adherence to these criteria ensured that each site was conducive to the evaluation of both product performance and support operations.
3. A controlled introduction steering committee that augmented the existing local sales and services team. This committee monitored all pre- and post-cut activities and ensured that all activities proceeded on schedule and according to plan.

At the beginning of this program, marketing staffs in six major U.S.

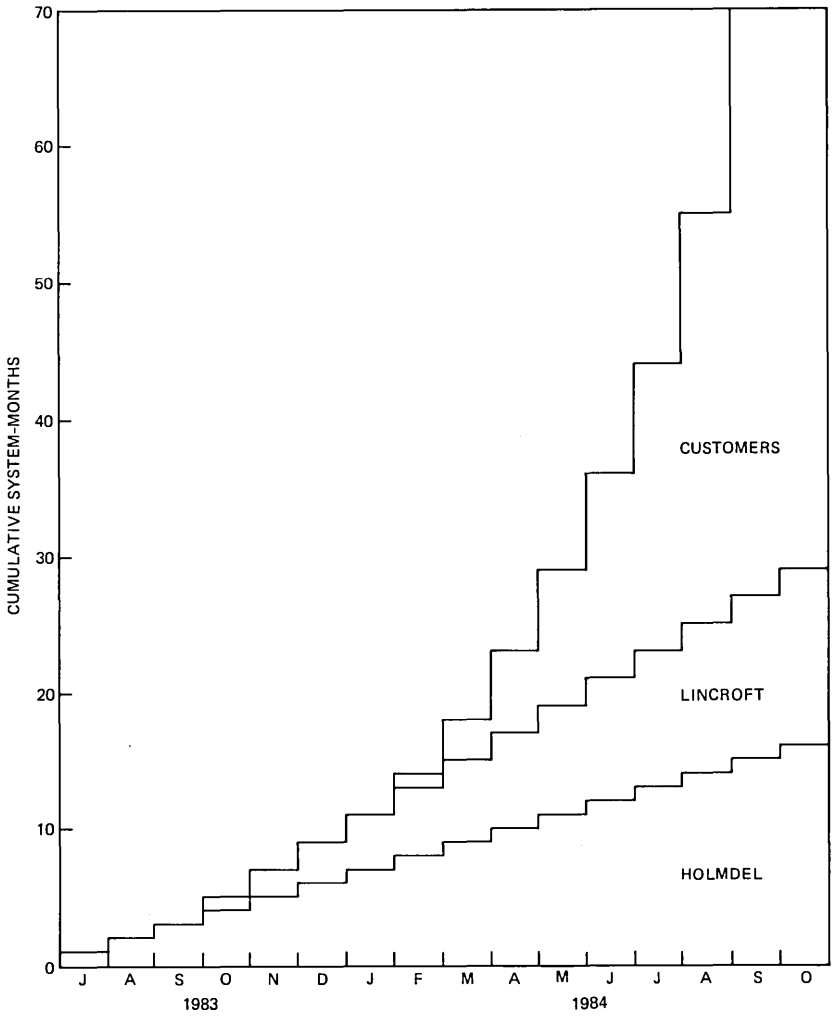


Fig. 1—Cumulative system-months of operation during controlled introduction.

cities were asked to review all their under-320-line accounts and submit profiles for those customers that matched the selection criteria. The list of potential customers was filtered and prioritized by the controlled introduction committee, which produced a list of accounts to contact. Potential customers were contacted by local sales teams, who described the aspects of being a controlled introduction customer. If customers were interested, the negotiations continued, a nondisclosure agreement was signed, and more details of System 75 were discussed. Once a customer requested that a configuration and contract be prepared, a team of product experts from both development and

Table I—Customer selection criteria

Customer Characteristics
First customers will be "smaller," i.e., 50–125 lines
Noncritical (e.g., no hospitals)
Weekend cut; 24-hour access; space for field-support equipment
No new buildings; no moves; equipment room complete
Potential use for wide range of product capabilities
Consider customers of past studies
Should be willing to participate in system management
Deploy a variety of size ranges for lines and trunks
An electromechanical PBX or Centrex replacement
At least one applications processor
Customers with need for Automatic Route Selection (ARS), and station message detailed recording
One System 85 customer
Market Environment
Friendly and flexible customers
Expectations should equal product capabilities
Leaders in their market segment
Large multiple-sales potential

marketing reviewed the proposed equipment order and the customer's intended use of the system. The expert team greatly enhanced the local support by answering detailed questions in an accurate and timely manner. This support is required to introduce products quickly.

III. TRAINING

One of the first activities required during the System 75 introduction was the development and refinement of training techniques and material. In particular, the preparation and delivery of quality customer, sales team, and service technician courses was a major step in assuring a successful product introduction.

3.1 Customer training

Two key ingredients made the customer training program successful. First, similar courses, especially those used for System 85, were reviewed for content and flow. These ideas provided the basis for the System 75 course. And, hands-on sessions with an operational system during the course development allowed the developers to try the material themselves and refine it. The first complete course was given to Lincroft users in September 1983. The first session for external customers was given in early December 1983.

During this course development, improvements were made in several major areas. One, the initial material did not contain enough detail. The customers wanted to know exactly what steps, visual indications, and audible indications were going to occur when a particular feature was used. Two, early material did not describe System 75's data switching capabilities. A thorough description is now included. Three,

feature descriptions alone did not usually convey their value. Initially, a more academic, logical flow method was used to describe features, but students reverted to using their own situations to verify their understanding. Once common business scenarios were presented, students quickly grasped the feature operation *and* its value.

And, finally, each student group seemed to have a focused interest, yet each group's interest was different. This called for a course that was broad based but customizable during each session so that each group could meet its own needs. A problem that is still under investigation is how to package the course material to allow for multiple customers to attend the same course and still allow a unique focus by each customer.

3.2 Sales team training

To ensure that local sales teams had sufficient System 75 product knowledge well before the time of announcement, preparation for sales team training began in November 1983 with the definition of a three-phase program. Representatives from the expert support team and from marketing worked in conjunction with the corporate training organization to develop and deliver the initial course, which covered voice features, data switching, system management, and system architecture. The first phase consisted of one week of lectures and lab sessions for area sales staff. The goal of this phase was to prepare the area staff to conduct their own training sessions for individual sales branch locations in their area. Some important attributes of this training phase were the use of working systems to allow hands-on laboratory sessions, the presence of product experts from the design team, the joint presentation of sales and technical information, and the chance for the sales force to give direct product-related feedback to development representatives.

The hands-on sessions were critical to the success of the entire training effort and greatly enhanced the sales teams' confidence. Aspects of system design or operation, which were difficult to describe in class, were easily clarified by demonstration. In addition, topics not covered in or triggered by the lectures could be easily discussed with the development representatives during the less formal lab sessions. Their expertise provided an added degree of credibility to the material.

The second phase provided three days of training to key representatives from each branch location. The area staff personnel who were already trained then combined with these key representatives to conduct one-day sessions for all the sales personnel at their respective branches.

In the third phase, trainers from the national sales training center visited the largest branch locations to conduct additional lectures and

host question and answer sessions. These last two phases enabled several thousand sales people to be trained quickly.

3.3 Service technician training

Development of the Service Technician training course began in April 1983. As with customer training, lectures and hands-on work sessions were interwoven to form a five-day course.

The first course was given in late 1983 to ten technicians. To verify their training, the first group of technicians helped install the Lincroft system the week immediately following the first course. This installation work identified the need to expand the cross-connect and wiring information in the course. Feedback from subsequent training sessions highlighted the need to improve the built-in diagnostic descriptions, the need to spend more time on feature operation, and the value of hands-on sessions. In general, the hands-on sessions stimulated more interaction between the student and the instructors, which in turn increased the students' understanding.

The final version of the course was completed in mid-1984. In addition to incorporating the suggested course enhancements, all its material was thoroughly checked for accuracy. Results from the field indicate that the training has been very effective.

IV. INSTALLATION

Another crucial controlled introduction activity was determining if System 75 could be installed properly and efficiently. The major elements of the system installation process were well known prior to the start of the controlled introduction. What was required was a way to identify and clarify all the minor aspects of a complete System 75 installation and fix any shortcomings that could delay the completion of an installation.

4.1 Internal sites

The first system to provide such information was a test system installed in Holmdel, New Jersey, during early 1983. This system was an early hardware prototype and contained preliminary feature designs and very limited administration and maintenance capabilities. System functionality as well as the number of users was gradually increased during 1983 and early 1984 as the design became more complete. Because the installation work was completed by developers, this system provided limited information on the minor details of the installation process. However, a second System 75, one that was near production quality and functionally complete, was subsequently installed at the AT&T Information Systems Laboratories in Lincroft,

New Jersey, during late 1983. This installation was more formal and followed standard procedures. The installation of these two internal test systems uncovered a number of missing but very important installation methods, tools, and system piece-parts. Specifically, the experience at these first two sites highlighted several areas needing improvement.

One was the need for a simple checklist that structured the major tasks. This checklist was at first thought to be unnecessary since the installation manual listed all the steps that were to be followed. However, the installers felt much more comfortable if they knew the overall sequence of the major activities, and were able to tell if they were "halfway" or "three-quarters" complete. Such a job aid is now available.

Second was the need for an improved method of keeping the cabling information such as room number and cable identification. During the second installation, it was quickly apparent that many copies of "home brew" forms were used to keep track of and cross-reference names, room numbers, jack identification, and cable numbers. During one part of the installation there were five different types of forms used to reference these data. By analyzing why people were using these various forms and studying their contents, a single form was developed that is now part of the built-in administration routines. Once the normal administrative data is entered into the system, the cable and jack information can be quickly added and only one sheet is needed for the cabling work.

And, finally, during installation there were occasions when only part of the translated equipment was installed. The absence of this equipment would raise alarms and cause confusion between "not connected" and "not working." By adding an alarm-hold command this confusion was eliminated.

4.2 Customer sites

The Services organization tracked various operational metrics associated with the installation of all the System 75 controlled introduction customer sites. The times to install the switch cabinet, wiring, cross-connect hardware, terminals, and labels, and central office connections were collected in a disciplined fashion. Actual times to install were then compared to the estimates used to develop the operations plans. These estimates were based on experience with similar products. Table II shows variations from predicated values for a few key installation activities and the overall total. Negative variation signifies that the actual times were less than the predicted value. Note that on average, the major activities and total times were close to the early estimates. Actual installation times will continue to be monitored, and

Table II—Variation of actual installation times from predicted times (%)

Major Installation Activity	Customer				Average
	C1	C2	C3	C4	
Switch	−24	+15	−35	+29	−4
Terminals	+26	+7	−34	−33	−9
Cross-connects	+43	−23	−61	+16	−6
Total Installation	+26	0	−42	−4	−5

those activities that are significantly above or below expectations will be used to readjust service force training and/or sizing needs.

4.3 Overall results

Throughout the controlled introduction over 50 installation procedures were modified to improve the process and minimize the installation hours. By closely tracking early installations, method changes were made before significant work-force hours were wasted.

V. MAINTENANCE AND PERFORMANCE TRACKING

The next important aspect of the controlled introduction was determining the effectiveness of the system's maintenance capability, its overall maintainability, and its quality and reliability.

5.1 Tracking

In late 1983, a field support group made up of System 75 product developers was established within AT&T Information Systems Laboratories to provide a close coupling between the field sites and the development community. Representatives from the field support group and a national services center combined to serve as backup to the local installation and maintenance technicians. In particular, they provided in-depth technical analysis. They also monitored the operation of each controlled introduction system using built-in remote-access capabilities[3] of System 75 as well as specially designed field support tools.[4] These field tools went beyond the system's diagnostic capabilities and provided more detailed information for sophisticated trouble-shooting. When design problems were identified, they were jointly investigated in the lab by developers and the field group. Solutions to field problems were verified first on lab models and system test machines and then soaked in the internal systems at Lincroft and Holmdel. Fixes were installed in customer sites only after the proper operation was verified at one of these internal systems. A modification-request process[5] was used to track all such problems. This process was completely on-line and could be accessed from the field or the lab.

This database allowed problems to be tracked by release, severity, community (such as hardware, firmware, software), and customer. Such tracking was very instrumental in coordinating fixes in an orderly manner.

5.2 Maintenance improvements

For every maintenance alarm, the maintenance action was recorded and identified as being either effective or not. This "maintainability" score card helped identify not only design flaws but documentation deficiencies as well. As a percentage, 75 percent of the maintenance actions that proved ineffective were due to documentation. Of these, some were due to inappropriate decisions to limit the detail in certain documents. The other 25 percent were actual functional shortcomings in the maintenance strategy and/or routines.

One example where a design change was required was in the amount of status information displayed during system initialization. The early design had moderate amounts of time between externally recognizable events (such as Light-Emitting Diodes [LEDs]* changing state or the tape starting or stopping). By having this system-access terminal display the internal status of the system while it was initializing, the technician could see that each initialization step was being completed.

All information about faults and maintainability discovered during the controlled introduction was used to expand training and documentation. For example, boards with typical field faults were used in the technician training course to demonstrate repair actions.

5.3 Quality and reliability

Detailed records of hardware failures were kept by the services organization. All defective circuit packs were returned to the hardware developers via the field support group for failure analysis. Information on packs with manufacturing defects was forwarded to the factory. The other defects were reviewed and design changes were implemented when needed. This activity continues throughout the life of the product on a special study basis. All results of failure analyses are coupled with factory repair information and used to compare actual field reliability to black box reliability estimates. These statistics highlight where corrective design action may be necessary and are used for inventory management. No design changes have been necessary to date because of reliability factors.

VI. MARKET FEEDBACK

The controlled introduction customers and sales teams provided

* Acronyms and abbreviations used in the text are defined at the back of the *Journal.*

useful product-related feedback. Their comments helped identify System 75's strengths as well as its areas for improvement.

6.1 Customers

Each controlled introduction customer, including the two internal sites, was asked to participate in a user survey. These evaluations were typically conducted four to six weeks after the system was placed into service. Survey questionnaires were distributed to about 40 percent of the user population. Face-to-face interviews were conducted with about 10 percent of the users. The questionnaire focused on training, documentation, and feature usage. A portion of the questionnaire is shown in Fig. 2. These surveys[6] indicated that

1. Overall satisfaction was high and System 75 was preferred over the customer's previous system.
2. System management emphasizing customer participation was considered the most valuable new capability.

III. FEATURE USAGE

For the features listed below, please indicate your responses by marking the appropriate boxes.

	(1) Do you have a button for this feature?		(2) How frequently do you use this feature?			(3) Is this feature easy to use?	
	Yes	No	Often	Sometimes	Never	Yes	Somewhat
FEATURE							
⊕ Conference	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Hold	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Transfer	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Drop	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Call Forward	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Leave Word Calling	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Cancel Leave Word Calling	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Send All Calls	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Call Pickup	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Automatic Callback	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Call Park	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Consult	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
⊕ Abbreviated-Dial	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]

Fig. 2—Sample questionnaire sheet.

3. All attendants were pleased with the console operation and design.
4. User documents were good but could be made less redundant.
5. Training was comprehensive.

6.2 Sales teams

Early interactions with sales teams and area marketing staffs were instrumental in identifying additional capabilities required of the product. For example, one group pointed out the need for line appearance bridging capability. Their insight into system usage was instrumental in adding complete bridging functionality to the system. In another case, a team asked whether or not calls to an active station could be conferenced onto its existing call. The answer was yes, but this fact and various uses of the capability were never stated in user documents or discussed during training. This triggered an addition to both the documents and the training course.

Currently, all product questions from sales teams are sent through the area staff to a national marketing center. This center has become another valuable source of product, documentation, and training needs data. These data are used to guide future product enhancements.

VII. CONCLUSION

In summary, the System 75 testing period spanned approximately 18 months. This included very early and limited service at internal locations followed by full service to 12 customers. Table III summarizes the major milestones. One of the most significant events was the System 75 installation at the Lincroft location. This site included an extensive executive complex as well as a broad range of other users. This provided both the opportunity to discover shortcomings and the pressure to fix them. Next in importance was the availability of an operational demonstration system in an appropriate display environment for training and early customer presentations. Without such a facility most of the training feedback mentioned earlier would have been discovered when it was much more costly to correct.

Table III—System 75 controlled introduction milestones

Date	Event
December 1981	Initial shipment and allocation proposal
November 1982	Controlled introduction plan issued
February 1983	Limited early prototype installed in Holmdel, N.J.
July 1983	Training and demonstration facility in Holmdel, N.J.
October 1983	Full-capability System 75 installed in Lincroft, N.J.
February 1984	First customer system installed
April 1984	Press announcement
May 1984	First system with applications processor installed

Throughout this process of testing, significant enhancements and corrections have been made to the System 75 design and its support methods. While much of the data collected was subjective, it was timely and sufficient to improve the product. By blending it with experienced judgment, System 75 was brought to the marketplace quickly and at the same time fully tuned and ready for rapid manufacturing buildup.

VIII. ACKNOWLEDGMENTS

The AT&T Information Systems Services Division provided details of performance and service tracking operations.

REFERENCES

1. A. Feiner, E. J. Rodriguez, and C. D. Weiss, "System 75: Introduction and Overview," AT&T Tech. J., this issue.
2. J. L. Gavegan, A. P. Ryan III, and G. E. Saltus, private discussions with J. A. Miller and J. J. Shanley.
3. K. S. Lu, J. D. Price, and T. L. Smith, "System 75: Maintenance Architecture and Functions," AT&T Tech. J., this issue.
4. T. J. Pedersen, J. E. Ritacco, and J. A. Santillo, "System 75: Software Development Tools," AT&T Tech. J., this issue.
5. T. S. Kennedy, D. A. Pezzutti, and T. L. Wang, "System 75: Project Development Environment," AT&T Tech. J., this issue.
6. D. E. Gordon, "System 75 User Survey Revisions," internal publication.

AUTHORS

Michael A. McFarland, B.S. (Electrical Engineering), 1970, Union College; M.E. (Engineering), 1971, Cornell University; Bell Laboratories, 1970–1977 and 1979–1982; AT&T General Departments, 1977–1979; AT&T Information Systems Laboratories, 1983—. Mr. McFarland has developed quality assurance audits for transmission systems and has been engaged in product introduction and project management activities for T1C and T1D carriers. He was based in St. Louis from 1979 to 1982 and provided liaison between Southwestern Bell and AT&T Bell Laboratories on a wide range of technical matters. In 1982 he began coordinating System 75 product introduction activities. He currently supervises a group that is evaluating marketplace reaction to System 75 and defining future system capabilities.

John A. Miller, B.S., 1966, Gonzaga University; M.S. (Electrical Engineering), 1967, and Ph.D. (Electrical Engineering), 1971, Stanford University; Bell Laboratories, 1966–1982; AT&T Information Systems Laboratories, 1983—. From 1971 to 1976, Mr. Miller worked on a variety of exploratory projects leading up to the development of the *Horizon*® communications system. From 1976 to 1980, he was Supervisor of the *Horizon* system Maintenance and Field Support group. Since 1980 he has been Department Head of the Support Systems department, responsible for project coordination, system test, and field support for System 75. Member, IEEE, Sigma Xi.

ACRONYMS AND ABBREVIATIONS

ACD	automatic call distribution
AMDT	ASCII message definition table
APEX	angel processor executive
AUDIT	data audit MAP
BM	board manager
BORSCHT	battery feed, overvoltage protection, ringing, supervision, codec, hybrid, testing
CAS	centralized attendant service
CCMS	control channel message set
CM	connection manager
CMTS	Change Management Tracking System
CO	central office
COM	communications manager
CP	call process
DCP	digital communications protocol
DCS	Distributed Communications System
DDD	direct distance dialing
DIP	dual in-line package
DLI	digital line interface
DMA	direct memory access
DPM	dial plan manager
DRAM	dynamic random access memory
DSP	digital signal processor
DTMF	dial tone multifrequency
EDC	error detection and correction
EIA	Electronic Industries Association
EPF	electronic power feed
FCS	frame check sequence
FSS	file system server
GAMUT	an automated testing tool
HDLC	high-level data link control
HIC	hybrid integrated circuit
HMM	high-level maintenance manager
HWMAP	hardware MAP
INITMAP	intitialization MAP
ISA	Information Systems architecture
ISDN	Integrated Services Digital Network
LAN	local area network
LATK	local administrative tool kit
LED	light-emitting diode
LSB	least significant bit
LTM	leisure time manager
MAP	maintenance action processes

MBus	memory bus
MCP	maintenance command process
MDM	maintenance data manager
MESA	management environment for software administration
MET	multibutton electronic telephone
MMU	memory management unit
MPLPC	multiple pulse linear predictive coding
MR	modification request
MSB	most significant bit
MSG	message service
MSTS	Milestone Schedule Tracking System
NM	network manager
NPE	network processing element
OGS	object generation system
OPTICS	Oryx/Pecos test, inquiry, and control system
PBX	private branch exchange
PCM	pulse code modulation
PCS	personal communication services
PD	project document
PM	process manager
PROM	programmable read-only memory
RAM	random access memory
RMATS	Remote Maintenance Administration and Traffic System
SAKI	sanity and control interface
SAT	system access terminal
SBus	system bus
SCCS	Source Code Control System
SCD	switch control channel driver
SD	service dispatcher
SIP	single in-line package
SMAL	structured macro assembly language
SMDR	station message detail recording
SPE	switch processing element
SSV	station service
TC	terminal controller
TD	time division
TDM	time division multiplexed
TS	time slot
TSI	time slot interchanger
TTL	transistor-transistor logic
UM	user manager
UL	Underwriters Laboratories
VLSI	very large-scale integration

AT&T TECHNICAL JOURNAL is abstracted or indexed by *Abstract Journal in Earthquake Engineering, Applied Mechanics Review, Applied Science & Technology Index, Chemical Abstracts, Computer Abstracts, Current Contents/Engineering, Technology & Applied Sciences, Current Index to Statistics, Current Papers in Electrical & Electronic Engineering, Current Papers on Computers & Control, Electronics & Communications Abstracts Journal, The Engineering Index, International Aerospace Abstracts, Journal of Current Laser Abstracts, Language and Language Behavior Abstracts, Mathematical Reviews, Science Abstracts (Series A, Physics Abstracts; Series B, Electrical and Electronic Abstracts; and Series C, Computer & Control Abstracts), Science Citation Index, Sociological Abstracts, Social Welfare, Social Planning and Social Development,* and *Solid State Abstracts Journal.* Reproductions of the Journal by years are available in microform from University Microfilms, 300 N. Zeeb Road, Ann Arbor, Michigan 48106.


AT&T