IBM Application System/400™ Technology

SA21-9540-0

Dedication

This publication is dedicated to the most important element of business: people.

To the people who purchase IBM products, our customers:

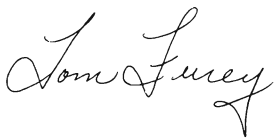
who displayed their enthusiasm for the S/3X product family and their interest in providing input and feedback that helped us meet and, we hope, exceed their product expectations.

To the people of IBM around the world:

who have dedicated their time and energy to develop the AS/400 product family. To employees in IBM manufacturing locations in the United States, Europe, Japan, and Mexico, who helped ensure that worldwide product requirements were met.

To the people of IBM Rochester and Toronto:

who have provided their ingenuity, commitment, and dedication, to deliver to our customers products with improved cost performance, the highest quality, and the most advanced computer architecture in the world.



Tom E. Furey, Jr.
Laboratory Director
IBM Application Business Systems
Rochester, Minnesota

Foreword

The AS/400 system is a new generation of general-purpose, mid-range systems from IBM. With more than 1/4 million customers worldwide, the product family that established the small and intermediate business computing standard is now setting a new standard with the AS/400 system. It has been designed and built to combine the strengths of its predecessors. This includes the System/36's large application portfolio and wide range of connectivity options, and the System/38's programmer productivity, advanced architecture, and integrated data base.

Significant new function has been added to enhance ease of use and connectivity and to support IBM's Systems Application Architecture (SAA), online education, and direct electronic customer-to-IBM support. The AS/400 system has been designed to provide growth potential for future applications, including applications with graphics, voice, and image capabilities. The architecture also preserves customers' application and education investments by providing easy migration for most applications. The hardware, with its expanded range, is setting new standards in quality and reliability while using the latest in IBM's VLSI, main storage, and disk technology. This has all been accomplished in a hardware family managed by a single operating system.

Such an undertaking provided many programming, engineering, and manufacturing challenges during development. This publication is a collection of articles on the design and development of the AS/400 system. The *AS/400 System Overview* provides a high-level look at some of these advances, followed by three main sections: Programming, Engineering, and Manufacturing. These articles were written by 78 of the more than 2,000 technical and professional people who work in these areas. They describe some of the innovation, technology, and design, and thus some of the advantages, built into the AS/400 system.

James E. Coraza
Director of Advanced Systems
IBM Application Business Systems
Rochester, Minnesota

Preface

The articles in this publication are not intended to replace IBM technical manuals in describing the capabilities of the system components and how to use them. The articles are for general technical communications purposes only; they do not represent an IBM warranty or commitment to specific capabilities in the referred-to products. These articles describe the AS/400 system at the time of announcement and will not be updated.

The publication is the result of the efforts of many people, but special acknowledgement is due to: Vicki J. Gervickas, editing; Michael L. Horsman, graphics; Wayne A. Larson, engineering; Richard J. Lindner, programming; Frederick R. Oeltjenbruns, manufacturing; John C. Kaplan, system support; Roger L. Taylor, system development; and Herbert B. Michaelson, publications consultant. They provided significant help with photography and graphics, as well as shaping both the publication and the individual articles.

Ben R. Persons
Managing Editor

First Edition (June 1988)

The information herein is subject to change before the products described become available.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

The numbers at the bottom right of illustrations are publishing control numbers and are not part of the technical content of this manual.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to your IBM-approved remarketer.

This publication could contain technical inaccuracies or typographical errors. Address comments concerning the content of this document to IBM Corporation, Information Development, Department 245, Rochester, Minnesota, U.S.A. 55901. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

On the Title Page: Models B10 through B60 of the AS/400 System.

Table of Contents

AS/400 System Overview

Provides an overview of AS/400 underlying concepts and a high-level view of technical innovations incorporated in the system.

Ronald O. Fess, Kenneth R. Reid, C. David Truxal, and Richard J. Lindner

Introduction

The AS/400™ system is a broad new family of general purpose mid-range systems. The system architecture was developed to provide a total solution to computing needs. It employs today's hardware technologies, combined using state-of-the-art engineering processes, to create a family of models tailored to a broad range of business needs. This system sets new standards for usability, performance, reliability, productivity, simplicity, and training, while offering solutions that allow it to grow with the needs of a business. The entire family is managed by a single operating system that provides complete end-user consistency and application portability across all models.

The AS/400 system provides many integrated features that form the foundation for a productive and extendable computing system. Operating System/400™ (OS/400™) provides a comprehensive, fully integrated set of batch and interactive work management functions that make processing application programs efficient and productive. Business operations are complemented by the integrated office products and sophisticated communications components of this system, which effectively employ attached personal computers and other systems within a network to provide maximum data availability. The OS/400 data management facilities provide a full range of data description capabilities and a consistent interface for application access to data. All data resides in a single integrated relational data base, with powerful query features that make information readily available. These facilities, combined with a range of high-level language compilers and utilities, provide customers with a set of highly productive application development tools. System utilities and system management facilities, such as message handling, spooling, and diagnostic support, make operating the system convenient and easy to understand.

Moving from predecessor systems to the AS/400 system is also convenient. Within OS/400, environments are available for easy migration of most System/36 and System/38 applications, files, and procedures. These environments make the applications appear as if they are running on System/36 or System/38, thereby preserving the customer's previous investment in applications and in the training to use them. This means an extensive application base already exists to meet business needs, while additional applications are being developed to capitalize on AS/400 advantages. The system also allows gradual conversion of existing applications to take advantage of the advanced AS/400 system capabilities as the user's business needs dictate.

Many software, microcode, and hardware innovations, plus the IBM strategic Systems Application Architecture™ (SAA™), make the AS/400 system a product for the future. SAA conformance will allow application movement to and from other conforming systems, with application users shielded from the underlying hardware and software differences.

System Concepts

The AS/400 system is designed and built as a total system, integrating IBM hardware and software components to provide optimal usability, performance, and reliability while reducing costs. Three basic system concepts form the underlying structures that give this system its advanced characteristics. The first is the **layered machine architecture**, which isolates the effects of change and makes the system function extendable in a manner transparent to the end user. The second is its **object orientation**, which permits an instruction interface that is consistent across a wide range of supervisory and computational instructions. This allows the operation and use of machine resources through logical names, independent of the underlying hardware specifications or characteristics. The third concept is the **single-level addressability** of all storage. This allows transparent storage addressing, making both main and auxiliary storage appear contiguous. This, coupled with its object orientation, allows dynamic object creation, use, and extendability, and permits storage and disk additions without affecting customer applications.

Layered Machine Architecture

The AS/400 system insulates users from hardware characteristics through the layered machine architecture. This layered architecture raises the level of the machine interface, creating a high-level machine instruction set that is independent of the underlying implementation. Figure 1 shows the hardware with horizontal and vertical microcode layers that comprise the high-level machine. The horizontal microcode (HMC)

layer implements the internal hardware instruction set, while the vertical microcode (VMC) implements the machine interface (MI) architecture of the system. The high-level machine provides many of the basic supervisory and resource management functions traditionally found in operating systems. Microcode runs faster than higher-level programs, so applications realize this gain when functions they use are implemented in microcode. The high-level machine provides the user with full multitasking, integrated security, automatic paging of programs and data, interlanguage calls with dynamic binding, interactive debugging, addressability for 2**48 (281 trillion) bytes of storage, and re-entrant programs (all users share a single copy of the program instructions). The machine interface allows implementation flexibility below it; therefore, machine performance and resources can be optimized by implementing function within the layer in which it operates most efficiently. Many validity and authorization checks are done by the machine microcode so software components can make simplifying assumptions about their input and operational requirements. As new hardware and software technologies emerge, they can be employed without affecting applications.

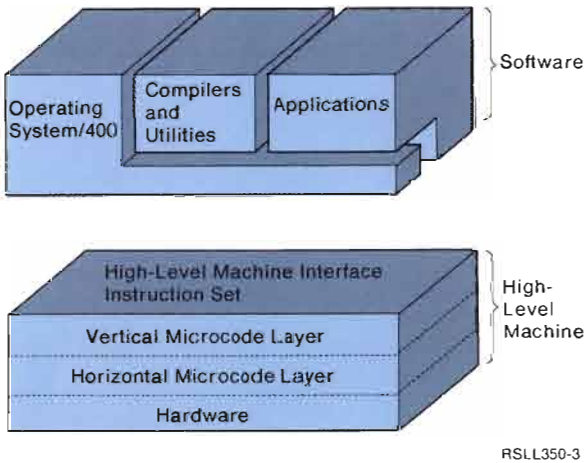


Figure 1 AS/400 Layered Architecture

Object Orientation

Everything on the system that can be stored or retrieved is contained in an object. Objects exist to make users independent of the implementation techniques and addressing structure used in the machine. The high-level machine is designed to treat everything the same through the use of a generic object structure. Although programs are run, files are read or written, and queues communicate results between processes or between a process and a device, they all have many of the same basic needs, such as storage space and security protection.

All objects are structured (as shown in Figure 2) with a common object header and a type-dependent functional portion. This permits the system to perform standard object-level functions on all objects, as well as permitting each object to be tailored for its own purposes. A create instruction is used to produce each object in a standard format. This process is called encapsulation and, once created, the object's internal format is not apparent to the user. The only exception to this is the space object, which is designed to allow storage of and operation upon byte-oriented operands such as character strings and numeric values. Object management functions ensure objects are used correctly, preventing inadvertent modification and ensuring they are available for subsequent operations.

With the object as the container for all stored data, machine interface instructions can handle everything in a consistent manner. Each object has an object-type identifier that determines how it can be used when retrieved. Complex system components combine several types of primary objects to provide composite objects. These composite objects are the constructs generally visible to the user; they are easier to understand and control because the complexity is handled by the system. For example, a physical file is a user construct that is made up of a data space object

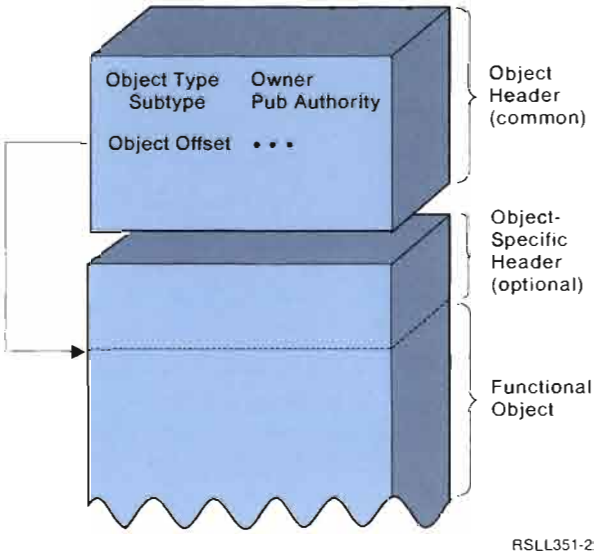


Figure 2 Structure of Generic Object

that stores the data, a cursor object that provides addressability into the data space, and optionally, a data-space-index object that provides logical ordering to records stored in the data space. By combining primary objects, high system quality can be achieved because proven functions are used, and system performance can be optimized by carefully tuning highly used functions.

Single-Level Storage

A single, device-independent addressing mechanism handles main storage and all auxiliary storage utilization. The system's directory contains virtual addresses rather than real disk locations. To run a program, the user simply calls the named program. This causes the high-level machine to automatically resolve the address and verify the user's authorization. The underlying virtual storage addressing mechanism ensures the program gets loaded into main storage and control is passed to the loaded address. In effect, the system branches to the program's address. Similarly, for data, a program considers data to be at its virtual address and the machine handles all

input and output operations transparently. For performance considerations, the system supports capabilities, called pointers, that contain an address and authorization for repeated access to an object.

Virtual addressing is completely independent of an object's physical location, and the type, capacity, and number of disk units on the system. A data base file may be stored by distributing it across several locations on several disk units, yet its records will have contiguous virtual addresses. As a result, multiple extent files appear to have just one extent, and unused spaces on a disk do not have to be gathered together to use them. Users have the advantage of being able to leave disk space management to the system.

AS/400 Software

The AS/400 system software contains advanced solutions for mid-range computer systems. These solutions are built upon the system concepts and their advantages are exhibited in many ways. Examples include the OS/400 user interface that makes the functions of the system visible and easy to use; AS/400 Office that provides a powerful environment for the control of business office operations, and integrates the use of personal computers; the system's advanced communications facilities that allow it to communicate with other systems in a business environment that often includes multiple office locations; and the operational control and system support capabilities that efficiently manage the system in a complex business environment.

OS/400 User Interface

Although many traditional operating system functions are performed by the high-level machine, OS/400 makes them easy to use. Simplified menus, commands, and help information are available to make the system easy to learn and make the user comfortable with its operation and control. A broad set of languages and utilities provide the support for effectively applying the system to business needs. Special functions make it easy to migrate from a System/36 or a System/38 to the AS/400 system. Figure 3 shows the relationship of operating system functional areas within the AS/400 system structure.

System functions are accessed through consistent, easy-to-use menus and commands with extensive online help and prompts readily available. Fast path (menu bypass) capabilities are also supported for experienced users. To support these capabilities an advanced User Interface Manager (UIM) is an integral part of OS/400. The

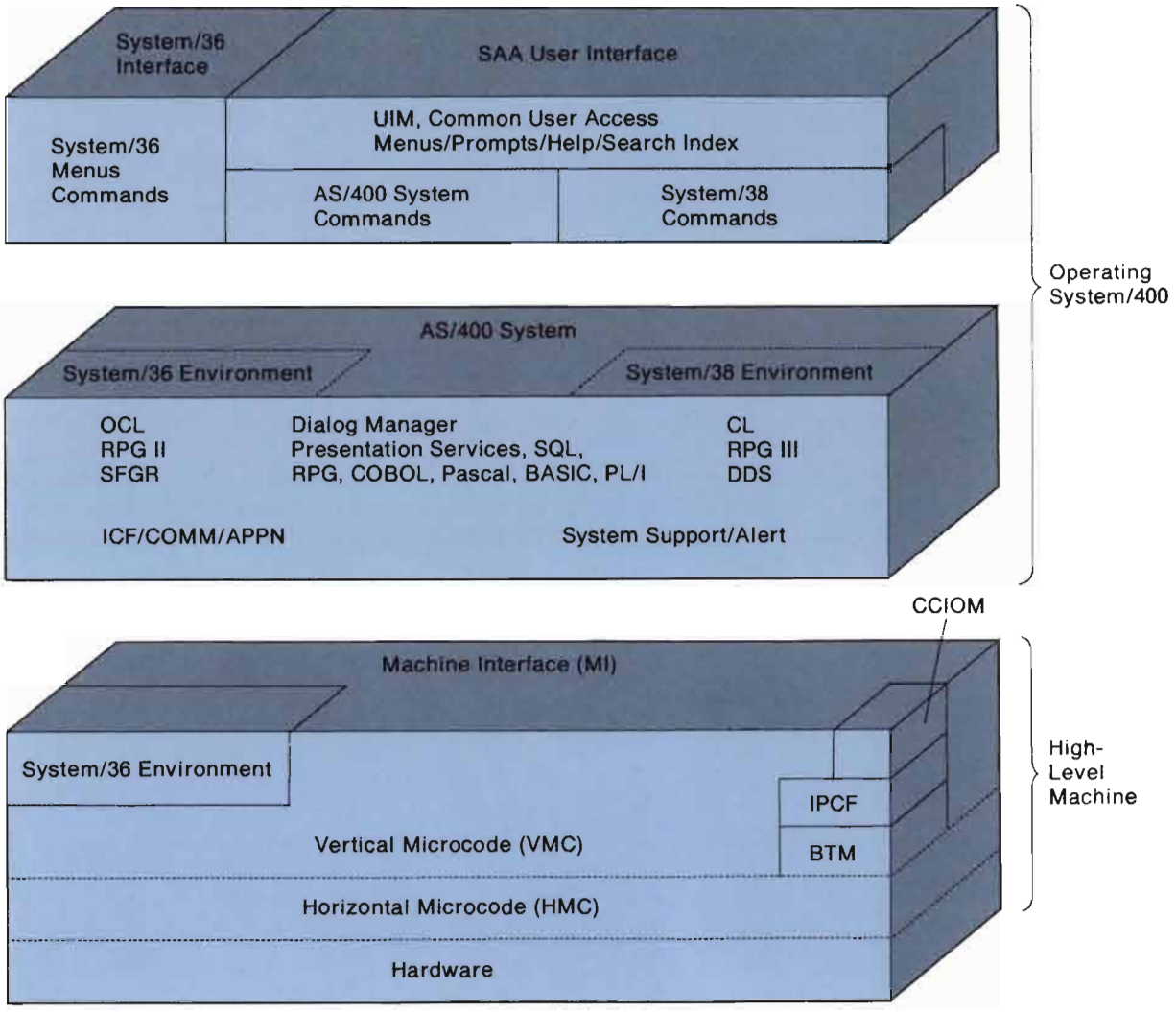


Figure 3 AS/400 System Structure

UIM is a device-independent display definition and presentation facility used by the system to provide a consistent interface, to enforce SAA interface standards, and to ensure consistent future user interface extensions. The system contains knowledge about functional dependencies; when an interactive request is made to the system, it uses that knowledge to tailor the prompts so only valid options are displayed. The UIM supports context-sensitive help information throughout the system. This help provides a consistent level of descriptive information that pertains to the field on which the cursor is positioned and is tailored to the task being performed.

OS/400 is designed to support interactive use in multiple national languages for worldwide application. Textual data is stored separately from operational program code, permitting a system to operate concurrently in many national languages. The UIM selects the appropriate language and shows online displays, messages, and help information, based on the user's profile object. The national language textual data can be updated or modified while the system is operational. Facilities used by the system to produce an international system are also available to customers for application development.

System facilities are available through a command interface. The comprehensive control language (CL) provides interactive users and application programs consistent access to system functions. This extendable, high-level interface has a consistent syntax for operational, programming, and maintenance functions. CL can be compiled to perform complex operational functions, and the resulting CL programs may call or be called from any high-level language program. They may also access the data base and they may communicate directly to users through displays and prompts. In addition to CL interfaces, the AS/400 system has provided various application program interfaces (APIs) that allow customers to tailor functions for their business needs.

Another interface is the system's Structured Query Language/400 (SQL), a strategic SAA interface to the system's integrated relational data base. SQL is integrated into the system using the same system services for security, locking, data storing, and retrieving that are used by all other AS/400 products. This allows the user to access the AS/400 data base with SQL language statements, utilities, or conventional high-level language read, write, and update statements. The system supports a comprehensive locking strategy at the object and record levels to allow multiple users to view, access, and modify the same files at the same time, through any interface, without losing data integrity. With this data base, the AS/400 system has the ability to access a single data representation using multiple logical views, providing advantages in application programming productivity, quality, and security.

Powerful data and file definition languages provide the external interfaces supporting data base, device, and communications files. File definitions, stored as objects on the system, provide the framework for independence between device files and data base files, allowing data to be optionally directed to a device or the data base through CL commands. The file definition languages describe data externally from the application programs. Once a file is defined, its definition can be used by many application programs. The data definition facilities were designed in conjunction with OS/400 data management, the high-level languages, and the high-level machine object-oriented functions. Together, these system functions give the application programs a very consistent data management interface. The high-level machine permits the system to perform typical application-data validity checking and formatting, thus improving performance significantly and providing more advantages in programming productivity and application quality.

Special support exists within OS/400 that allows applications to operate as though they were running on a System/36 or System/38, making it easy to migrate end users and most application programs to and from the AS/400 system. A System/36 environment exists that consists of a machine definition, commands, procedures, files, and a set of control blocks, which are added to a job when the associated user-profile object has a System/36 attribute specified. This attribute informs the system that it should operate in a manner consistent with a System/36. Users without the attribute in their profile can issue a CL command to initiate the System/36 environment. The System/36 environment runs like a subsystem within OS/400, not as an emulator; therefore, the user does not pay the performance penalty normally associated with an emulator.

Similarly, a System/38 environment supports running System/38 user applications. This environment is established when a program has the System/38 attribute. System/38 programs run as part of AS/400 jobs and use System/38 command syntax, command definitions, and function. This attribute is automatically established when an object is restored on an AS/400 system from a System/38, so applications can run without change. Objects can also be created with this attribute.

Office Environment and Personal Computers

AS/400 Office can help control the flow of work in an office. It is an integrated office services package that contains support for word processing (including text, graphics, and image processing), calendars, electronic mail, personal directories, and office administration. Its user interface includes full suspend-and-resume capability from all office functions, permitting

users to suspend any office function, initiate other functions, and later resume any previously suspended function. Customer applications called through the Office menu may be suspended or resumed as well. The office user interface is consistent with the rest of the system, including panel formatting and context-sensitive help throughout.

The AS/400 system structure and high-level machine have been used to optimize office performance and simplicity. The linguistic spelling features have been implemented in VMC for improved performance, and the support for electronic mail uses the networking support within the operating system and VMC, making the communications hardware complexity transparent to office workers. Office functions are designed in compliance with SAA guidelines using strategic IBM architectures. AS/400 Office contains a Document Interchange Architecture (DIA) library. The documents conform to the Document Content Architecture for future compatibility, and are distributed using SNA distributed services (SNADS). APIs are furnished with the system, allowing applications to use these office services directly.

Personal computers have been integrated into AS/400 Office through cooperative processing techniques, which distribute the work between the System Processor and the personal computer. Also, system support allows the AS/400 document library to serve as a filing system for the personal computer with data location made transparent to the PC application programs.

The method used to attach personal computers to the AS/400 system is transparent to the applications running on them. They can be attached by either twinaxial cable, synchronous data link communications (SDLC), or the IBM Token-Ring Network. A router that runs on the personal computer controls all communications to host AS/400 systems. PC applications behave consistently in all hardware environments. The router also allows the personal computer to have multiple sessions active with one or more host systems in the communications network. Office support, called work station function, running on the personal computer provides an interface for applications that wish to communicate with multiple host systems. Work station function also improves graphics performance by automatically requesting increased communications packet size, and improves graphics appearance by providing attributes to the host AS/400 system that allow it to tailor the graphics data stream to the display capabilities.

Cooperative processing techniques have also been used for host-dependent display stations, with text processing distributed between the System Processor and the work station input/output (I/O) processors.

Advanced Communications Facilities

Communications between systems is vitally important in the complex business environment of today, with the need to distribute information and have cooperative access with good control. AS/400 communications facilities give users the advantage of being able to communicate with other systems through automatic features that ease the management of complex communications networks. The key to this advantage is the advanced, extendable communications structure that provides independence from the specifics of communications protocols. This is important to the portability and simplicity of application programs. The AS/400 system achieves independence by integrating the industry's standard protocols into its communications structure and through the intersystem communications function (ICF) within AS/400 data management. ICF does the protocol-specific processing. Through the use of OS/400 data definition facilities and the communications function manager, it provides a high-level interface for communications.

ICF uses the advanced peer-to-peer networking (APPN) support, which is an extension of the Systems Network Architecture (SNA) networking function of the VMC layer. APPN provides automated functions like locating a remote resource, selecting the best data transmission route, activating a non-configured logical unit (LU) description, and automatically adapting the data transmission pacing and the transmission priority to optimize resources.

Automated functions are essential to communications usability. As the scope of business operation increases, the performance of the communications support also becomes critical. The protocol independence built into the AS/400 communications structure achieves a performance advantage through the use of separate I/O processors that relieve the System Processor from the compute-intensive burdens of communications data handling. The structure is extendable to accommodate the hardware and processing techniques of the future. As communications speeds and bandwidths increase, the benefits can become available to application programs without affecting their operational interfaces.

The interprocess communications facility and bus transport mechanism (shown in Figure 3) are new I/O architectures for data communications between the System Processor and the I/O processors. They operate in the VMC layer, and together make the transport mechanisms transparent to the communicating processes. The interprocess communications facility provides the logical connection between pairs of communicating processes, and the bus transport mechanism provides the transport services used by the interprocess communications facility to transport data across the I/O bus. The bus

transport mechanism is unaware of the content or format of the data. Device-specific or I/O processor-specific data structures are moved transparently, with processing left to the processes running in the System Processor or I/O processors.

Operational Control and System Support
The AS/400 system is shipped with the necessary user profiles, subsystem descriptions, device files, and other objects that make the system ready to use when installed. Additional tailoring of the system can be done at any time using the system menus or the command interface. Devices can be added without disrupting end users, using the concurrent configuration support to create the necessary configuration objects. Local devices can be attached to the system and automatically configured when they are powered on.

Online education is available, allowing users to learn at their own pace. These online courses, combined with the message help text and natural language, online search-index capability, are designed to provide education to users when it is needed, directly on their display stations.

The AS/400 system eliminates the need for separate resource management subsystems that impose functional restrictions and inconsistent interfaces on users. The high-level machine manages the flow of work and the allocation of system resources, supporting concurrent processing of batch, interactive, and transaction-oriented applications.

To protect data, security, which is the responsibility of the user, can be tailored to match user's needs or the controls to which the user is accustomed. A wide range of security levels are available, from inactive to full security, authorizing specific users to specific objects. Various degrees of user authorization can be selected between these two extremes. Security can be administered by specifying the list of users authorized to each object and the level of authority for each user. Or, it may be administered by specifying the list of objects authorized to each user and the level of authority the user has to each object. Features allow either method to be used, accomplishing the same end result. The machine performs all authority checking when an instruction first addresses a secure object. This gives excellent performance and transparency to application programs.

Advanced support facilities maximize availability of the system using sophisticated problem detection, isolation, and analysis techniques, and an innovative help network for commonly asked questions and answers. A set of electronic support functions have been integrated into the AS/400 system to help users identify problems. VMC and HMC support components record and report vital data about the hardware, software, and system conditions at the first indication of a failure. This information allows OS/400 functions to analyze and isolate a problem and electronically report it to the IBM service and support systems.

AS/400 Hardware
The AS/400 system hardware architecture complements the OS/400 software, efficiently supporting the high-level machine interface and system concepts. The engineering design employs the latest technologies, achieving dramatic performance and capacity improvements over predecessor systems, while providing competitively priced systems across the entire mid-range.

Figures 4 and 5 depict the AS/400 hardware structures. Two hardware structures are specifically designed to allow an extensive range of models and I/O devices that fulfill customer price and performance requirements. The smaller models are intended for use in a quiet office and fit inconspicuously in an office corner or beside a desk. The larger models are comprised of hardware components installed in a rack for expandability within a confined space. Together they provide an initial family of six models offering more than a seven-fold range in throughput performance. Main storage uses IBM's latest 1 megabit production storage technology. The disk storage offered on the AS/400 system features IBM's most advanced disk units where all track accessing and positioning is controlled by a separate microprocessor, thereby freeing the System Processor to do more application work.

The key elements in developing this broad range of hardware were the leading-edge processor implementation using very large scale integration (VLSI) logic, the storage and disk unit technologies employed, and the sophisticated development tools used to design and test the system. Just as OS/400 supports the SAA architecture, using common software products to increase application portability, the AS/400 hardware uses common hardware components and I/O microcode to provide OS/400 portability across the product family.

The System Processor communicates with independently functioning I/O processors over a high-speed bus for direct data access. Main storage access for the System Processor is provided by virtual-address translation (VAT) hardware that converts virtual addresses to main storage addresses. Address translation tables in main storage and a translation look-aside buffer in hardware provide high-speed assistance for mapping from virtual to real main storage addresses. The System Processor has a 32-bit data path and 48-bit addressing that can provide direct access to 281 trillion bytes of storage. It is implemented with a software and hardware architecture that can accommodate up to 64-bit addressing. If the future need arises, changing to the 64-bit addressing can be transparent to OS/400 and application software. This addressing

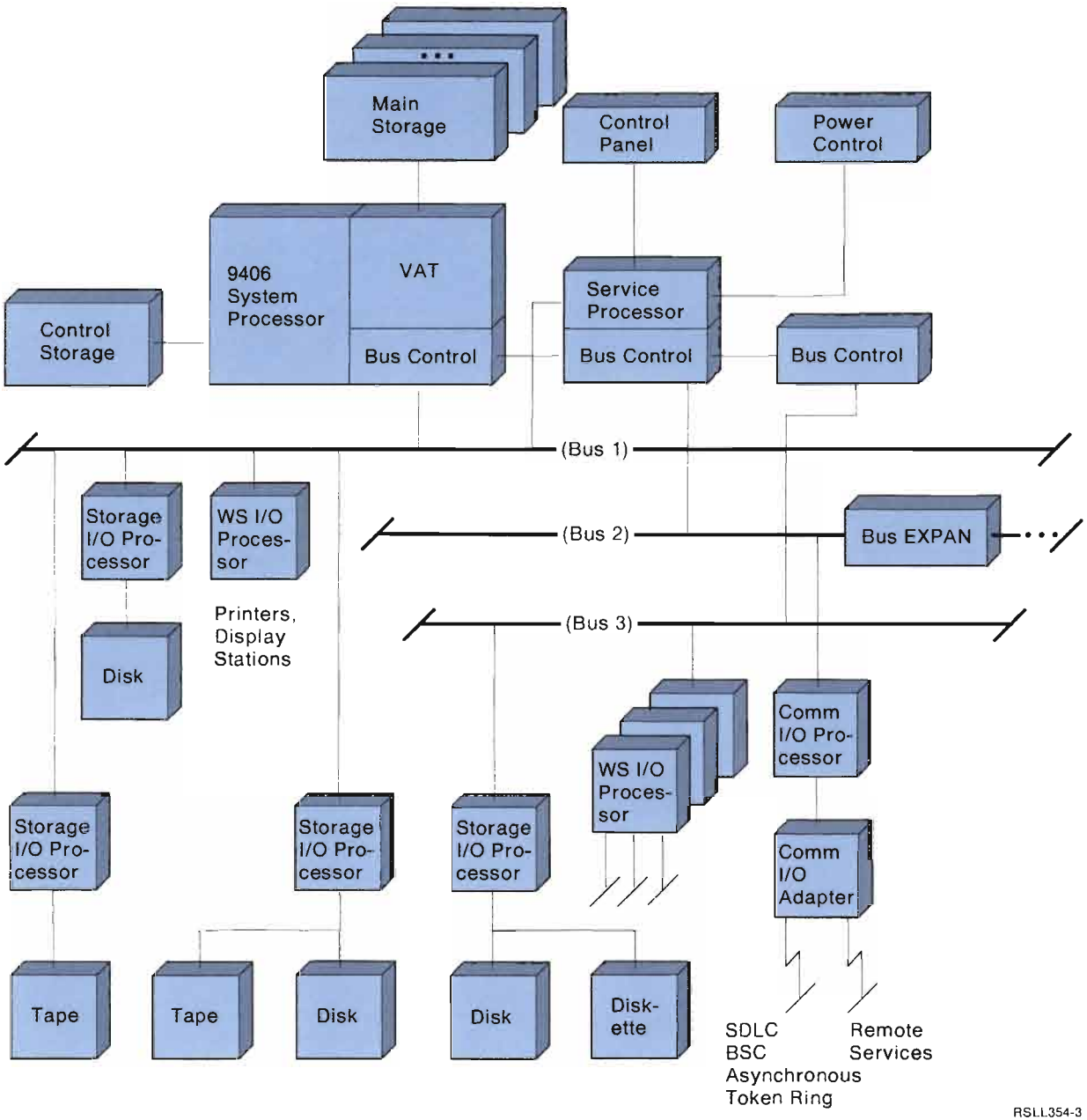


Figure 4 AS/400 Models B30-B60

model, coupled with single-level storage, provides the basis for independence from disk unit and storage characteristics and eliminates the need for disk management at the application level. While the industry implements 32-bit architectures, the AS/400 system's 64-bit architecture goes beyond to accommodate the needs of future applications with voice, image, and artificial intelligence capabilities.

The System Processor fetches 42-bit HMC machine instructions from the random access memory (RAM) control storage and can perform 8-, 16-, or 32-bit operations. Most machine instructions use one processor cycle. The processor cycles vary from 60 to 120 nanoseconds, depending on the model. The high-speed horizontal microcode provides good performance characteristics with a very flexible approach to creating VMC instructions. This layer of microcode isolates the rest of the system from the hardware characteristics. This will allow future performance, capacity, and cost improvements to the hardware without disrupting the operating system or customer applications. The use of microcode layers with built-in interfaces also permits movement of functions into high-speed hardware implementations as technologies advance.

Control storage is implemented in two sizes, 4K and 8K words. This allows trade-offs between cost and performance and is one of the techniques used to provide different price and performance options within AS/400 system models. Control storage is more costly than main storage but is significantly faster; therefore, the 8K control-storage system costs more than the 4K system but offers better overall system performance. In the low-cost 4K control-storage design, additional machine instructions are stored in main storage and the processor fetches these instructions from main storage rather than control storage.

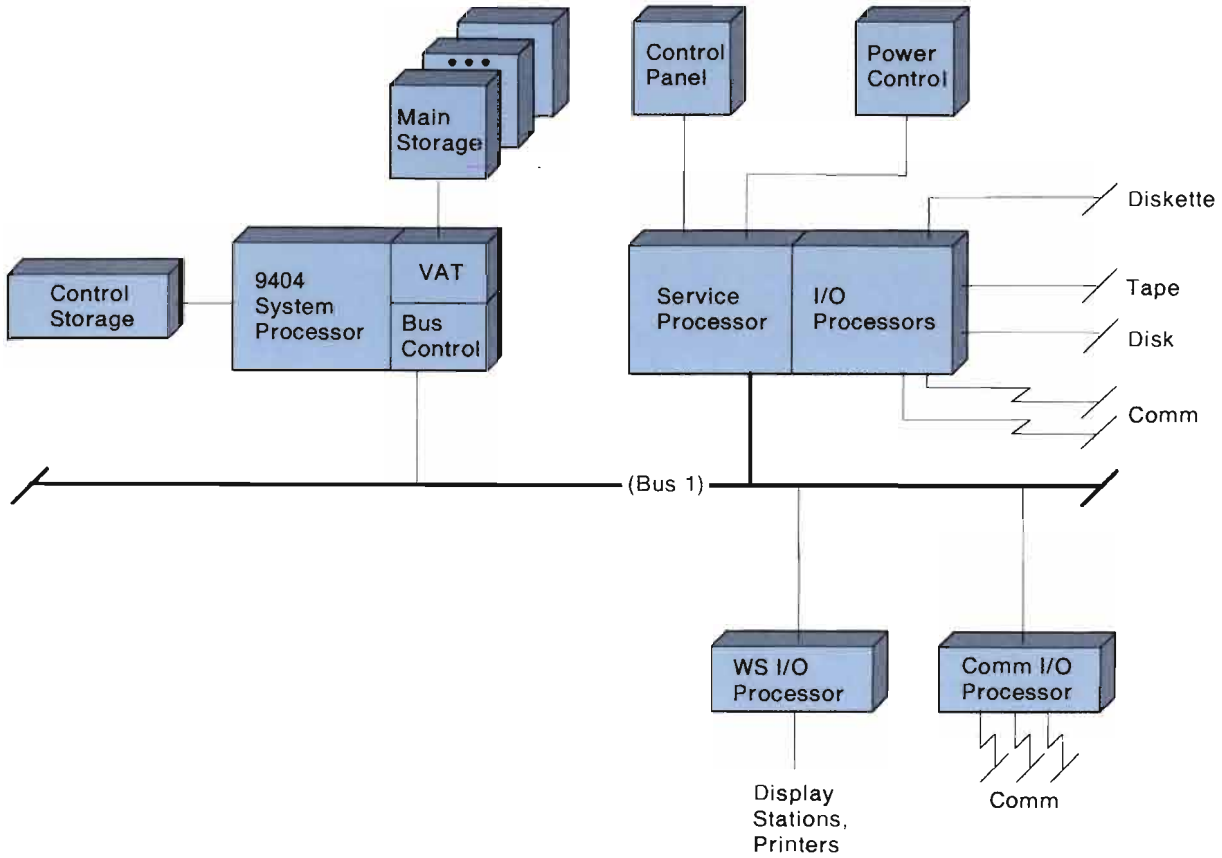


Figure 5 AS/400 Models B10 and B20

To optimize system throughput and response time, it was necessary for IBM to match technologies. The latest high-density storage technology is used as the basis for main storage and is available at two performance levels (80- and 120-nanosecond access times). The data path to main storage is 8 bytes in width, and capacities vary from 2 megabytes to 96 megabytes, depending on the model. In each model, processor speed is balanced against main storage and control storage access times and sizes to achieve optimal performance.

Each system also includes a Service Processor that starts the system and performs system maintenance functions, such as fault isolation, error detection, error reporting, and System Processor service utilities. The Service Processor also provides the interface to the control panel and power control functions.

To preserve IBM's traditional leadership in system reliability, availability, and serviceablilty, new methodologies were defined to analyze and implement hardware quality and reliability. Ease of service has been designed into the AS/400 system's online diagnostic capabilities and service publications. The system's functionally packaged hardware and electronic support features provide the capabilities for responsive service dispatching.

The larger AS/400 models are designed to provide maximum performance and capacities using dedicated I/O controllers for disk, tape, and diskette units, for communications and token-ring network lines, for locally attached work stations and printers, and for the Service Processor. These are shown in Figure 4.

The smaller AS/400 models are designed to provide the lowest-possible entry price, with competitive performance and capacity. The base configuration has I/O and Service Processor functions combined in one I/O processor. As additional I/O devices are required, individual I/O processors are added to support additional local work stations and printers or communications lines. These are shown in Figure 5.

Conclusions

The AS/400 system is IBM's richest, most competitive mid-range system offering. It provides broad function and a wide capacity range, with an advanced architecture and operating system that span an entire family of hardware. It features state-of-the-art data base capabilities, productive features for application developers, and distributed processing capabilities accessed through an easy-to-use interface that is based on the SAA Common User Access interface. It is an outstanding office system that integrates the personal computer with general-purpose systems.

This system preserves customer investment in education and most application programs, with environments that make most application source

portable to and from predecessor systems. The advanced facilities of OS/400, including sophisticated communications and networking functions, can be employed gradually as business needs require. Computer-aided problem diagnostics and online education advance the meaning of customer support.

Consistent implementation of the IBM SAA standards and products fortify this product family with strategic products today. The AS/400 system features hardware and software extendability to accommodate demanding applications of the future.

Programming

The programming of the AS/400 system has provided software advantages built on an advanced, extendable system architecture that supports the entire range of hardware models.

An Integrated User Interface

Describes a new dimension in user interface consistency and advancements that increase ease of use and simplify work activities.

J. Howard Botterill, Dennis A. Charland, and John Y. Harrington

Introduction

The AS/400™ system spans the range of small to intermediate systems. It addresses the needs of the single-user environment, as well as complex environments with many work stations and many users. The user interface is simple and self-guiding for new users, and is efficient and productive for professional data processing users. Although the system is new and advanced, the interface is based on the proven ease-of-use techniques and system-wide consistency of predecessor systems. It is a single integrated interface that combines the strengths of user-friendly menus, self-directing entry displays, extensive help, powerful list displays, a comprehensive command set, and an underlying object structure.

While retaining these proven techniques, the AS/400 system provides major enhancements resulting in greater ease of use, productivity, and flexibility. It expands interface consistency to include consistency with other IBM systems and between dependent work stations and attached personal computers.

The interface is designed to be flexible to address the broad spectrum of new and experienced users. This has been achieved by providing a primary method of interacting (usually choosing from a set of numbered choices) and alternative, fast-path methods for more proficient users. These alternative techniques, which include specifying multiple actions at one time, taking a direct path to any menu, and entering commands directly, can be used in combination with the primary method of interaction. They are designed so that users can easily graduate to them, using the same terminology, options, and order of specification as in the primary method. In this way, the interface grows with the user.

The menus provided with the system have also been improved. These menus allow the user to operate in action-object or object-action sequence, where the action identifies the task and the object is the item the user wants to operate on. (The object may not be an AS/400 object type.) After selecting the type of object (such as file, document, or program), users are presented with a list of objects of that type. On the list of objects, the user can type the number representing a desired action (like change or delete) next to one or more of the objects. The user can stay on that list of objects and follow that action with other action requests. As an added feature, a list display has a blank list entry that can be used when the name of the object is known. The user can type the action and the name of an object, without having to find the object in the list. In this same way, the user can create a new object by typing the number representing create and the desired name in the blank list entry.

While the list displays allow the new or occasional user to simply identify one action to be performed on an object, users, as they graduate to needing more function, can request actions to be performed on multiple objects. These actions can all be of the same type, or they can be different actions requested on different objects. When an action is requested that requires additional information, like the options for a print request, an entry display is presented requesting only the required and frequently used options. The more advanced, special purpose choices are available by pressing a function key. In this way, new or infrequent users are not intimidated by the full function. They only have to deal with the frequently used options that have meaning to them.

At any time, users can ask for assistance by pressing the Help key. Help is provided in the form of online text describing the field or display area the user is currently using. From that first help display, a function key can be pressed to get to a Search Index function. This Search Index function is a significant advancement. It allows users to request more information by supplying, in their own words, a description of what they want to know. In response, they receive a list of topics from the index that satisfies their request; from this list they can choose the ones they want displayed. In this way, the valuable tool of an index is automated by providing a word search into the help information that addresses the entire system.

These fundamental features of the AS/400 user interface allow users to initially use the system with little training, continue to use it occasionally, or become highly efficient users.

New Dimensions of Consistency

A consistent user interface is very important to the ease of use of any system. With the increase in networking and the use of personal computers, which allow the user to interact with different host

systems or the personal computer itself, has come the need for consistency beyond the individual system.

With the advent of the AS/400 system and Operating System/2™ (OS/2™) for the personal computer, IBM introduces new dimensions in consistency: consistency between different systems and between attached personal computers and dependent work stations. The set of rules and conventions that defines this consistency is called Common User Access (CUA) and is part of IBM's Systems Application Architecture™ (SAA™). Much of the CUA interface originates from the ease-of-use characteristics of the System/3X family of products [1], with enhancements to improve the interfaces on both attached personal computers and dependent work stations and make them more similar, without compromising the potential of the personal computer interface. CUA establishes IBM's direction for the future in terms of user interface and ease of use. (For more information on CUA, see the IBM publication on CUA [2].)

The AS/400 interface on both dependent work stations, like the 319x display stations and personal computers emulating them, and on the attached personal computers (provided by AS/400 PC Support) is based on CUA. This means that a personal computer will have the same function key assignments for common dialog functions, whether it is talking to an AS/400 system or functioning as a stand-alone personal computer. For example, F3 is used for exit, F4 for prompt, and F12 to return to the previous display on AS/400 work stations as well as IBM's new CUA-conforming OS/2 and System/370 products.

The AS/400 system and CUA consistency goes beyond function keys to dialog design and interaction. Dialog techniques such as single selection, value entry, list handling, and help are also consistent between the AS/400 system and the new personal computer products. Whether users are using OS/2 personal computer software or the AS/400 interface on a personal computer or a dependent work station, they can interact in similar ways. This is shown graphically in Figure 1. For example, if a user wants to select from several

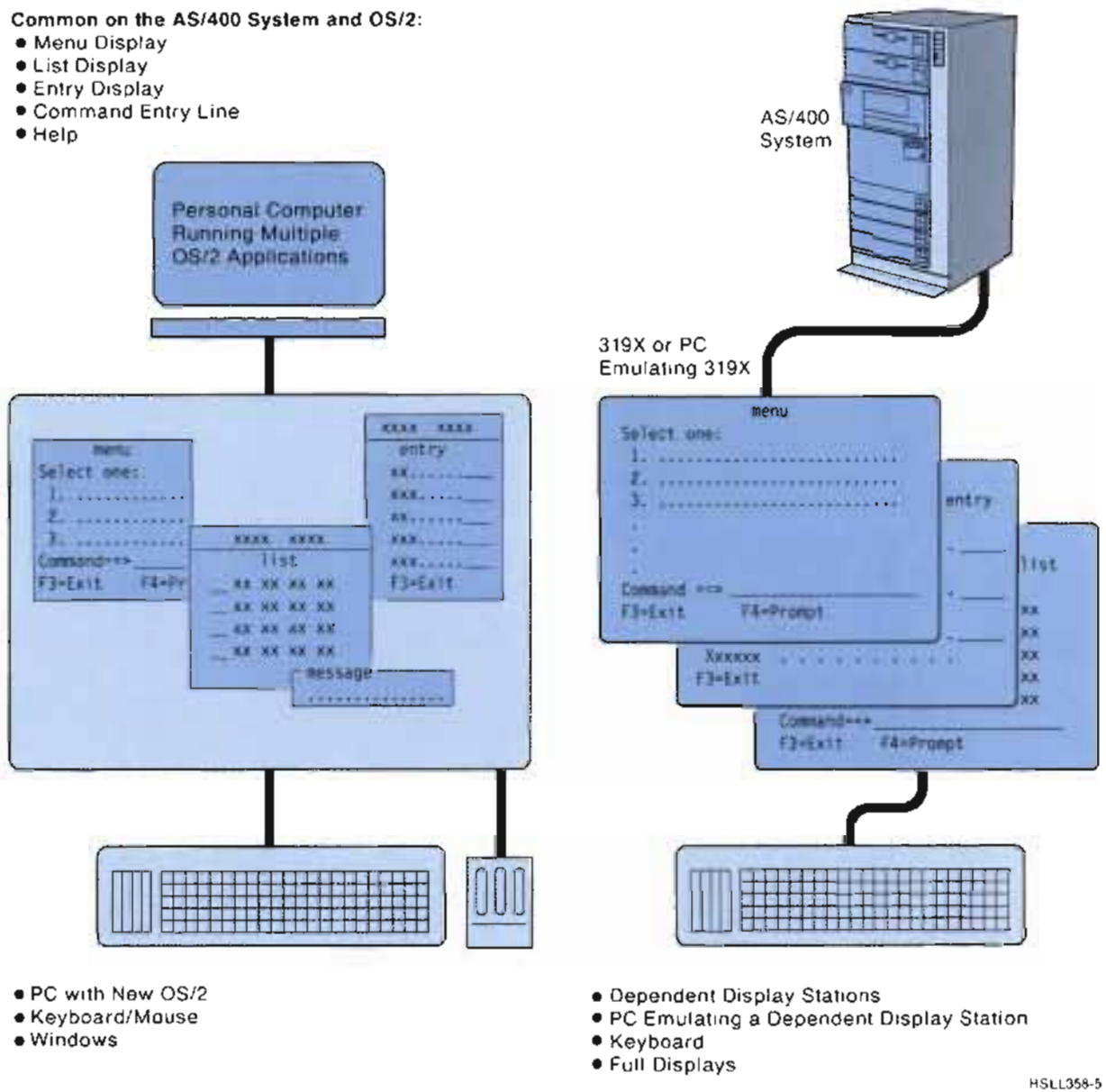


Figure 1 CUA Consistency Between the AS/400 System and OS/2

choices, the choices are presented in a common format in a window or full display. Users select their choice by typing its number or by pointing to it with the mouse. (A mouse is not available on the dependent display stations.) Similarly, the entry displays, information displays, and list displays are formatted and operate in the same way throughout the AS/400 interface, as well as being consistent with the appearance and operation of the same type of displays on other CUA-conforming products.

The resulting consistency between systems complying with CUA makes it possible for users to count on a common way of interacting, regardless of what system type of device is being used.

Menus

A comprehensive set of menus allows users to quickly identify and select the type of object to work with or the task to perform. The type of object may be a file, a document, a job, or mail, as shown in Figure 2. The task, for example, may be an office task, an application, a system operation, or a programming task. Some choices show lower-level menus, with a more refined grouping of choices.

In this way, the interface can accommodate either action (task) or object requests. Usually task choices go to a task-specific entry display. Object requests usually result in displaying a list of the requested type of objects to which the user is authorized. On the list display, one or more actions can be requested on the displayed objects. In either the task or object case, the user need not know any commands, keywords, or option names. Where needed, the system presents an entry panel with multiple fill-in-the-blank prompts. If a single choice is needed, a menu is presented.

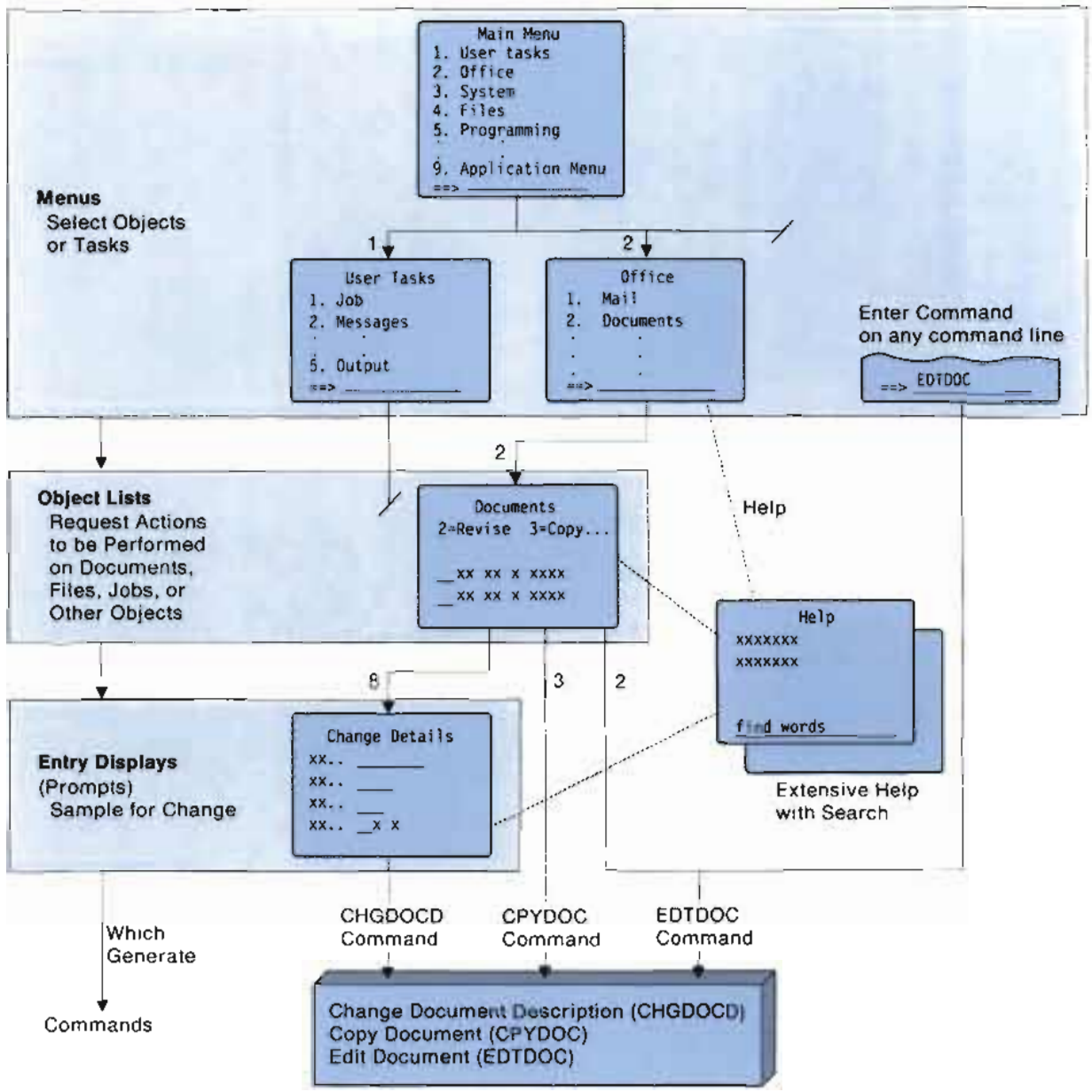
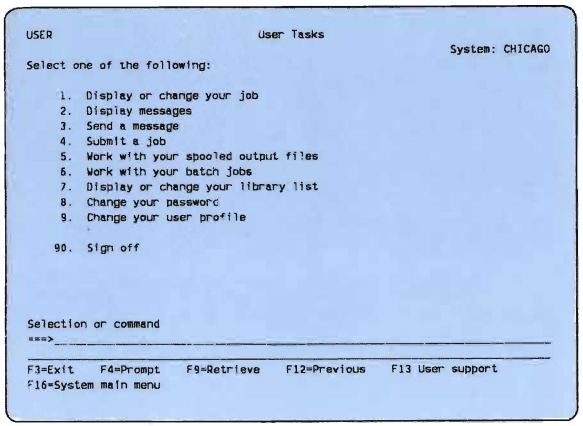


Figure 2 Integrated Display Interface

Specific menus are provided for common groups of tasks, such as office, programming, and operation. A new User Tasks menu is provided for users who are not data-processing professionals and do not need the full function of the other specialized menus (see Figure 3). For example, such users may use this menu to send a message to a co-worker (option 3) or check on their printed output (option 5) without having to be trained as a system operator.



RSLL360-2

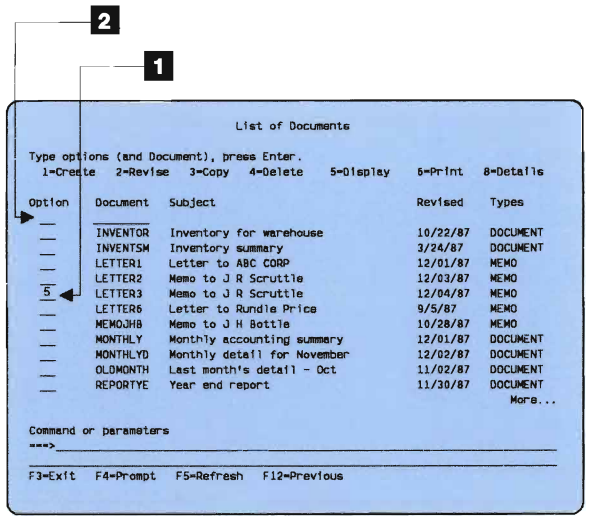
Figure 3 User Tasks Menu

A command line is provided on most system menus. Individuals who use the system frequently can display any menu by typing GO and the menu name. Other commands can be entered on the command line to request functions without using the menu option paths or leaving the current display. For example, EDTDOC entered on the command line of any menu (as shown in Figure 2) runs the Edit Document function.

List Displays

When a type of object is requested on a menu or by using a command, a list of objects, including type and attribute information, is displayed. (Figure 2 shows a menu request resulting in a list of documents.) A list display provides a convenient means to perform actions directly on objects, without having to recall and enter an object's name for each action. An action option field precedes each entry, and the action options supported are shown in the upper instruction area. Actions are requested by entering an option number in the field preceding the object. Figure 4 shows the list of documents with a 5 (**1**) typed next to LETTER3 to request a display of the content of LETTER3.

In the key areas of data definition, query, and office, the AS/400 system introduces enhanced list displays with an input-capable list entry at the top of a list (see Figure 4, **2**). This entry allows users to type the name of an object, along with the action option, without having to roll to the object or leave the list area. It also allows a request to be typed to create an object that is not in the list. The user can perform these actions in conjunction with other actions on objects in the list.



RSLL363-2

Figure 4 Example of List Display with Extended Entry

Entry Displays

Entry displays, which allow users to fill in the blanks, are provided when more details are needed after a task is selected. Figure 5 shows an entry display for a print request. The entry displays are straightforward and require a minimum of user interaction. They have a single column of entry fields, each preceded by a simple descriptive phrase (called a field prompt) and followed by a list or description of the acceptable values for that field. The values can be numeric values for fixed, non-command choices or actual command values, like *NONE, for command prompt-entry displays.

The user is asked to respond only to required and frequently used option choices. Default values are already entered in the fields. Choices that are only required in some situations are not initially presented. They are presented on a following display if it is determined, based on the initial responses, that more choices are indeed necessary. For example, if a copy request refers to a diskette file, only diskette-related options follow. Tape or data base options are not shown. This tailoring of the entry displays based on user responses is called **intelligent prompting**. The system tailors the prompts based on user responses. The displays are also layered. The fields that are less-frequently used because they are for advanced function are not initially shown. They can be requested by pressing F15 (Additional options). Each of these techniques results in users not having to analyze the individual fields or choices that do not apply to their task.

The fields on an entry display take two forms. The first form is an entry field, which requests a user-supplied value, like a name (see Figure 5, **1**). An underscore shows the value's maximum length.

For certain entry fields that accept a name, the system can show a list of the objects to which the user is authorized. **F4 for list** is shown to the right

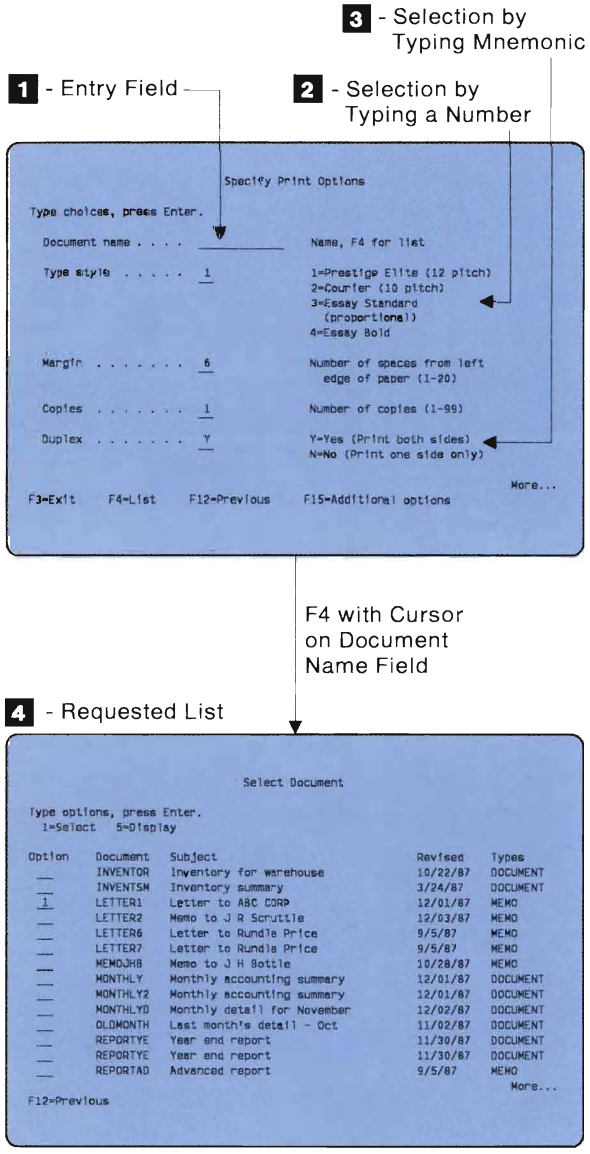


Figure 5 Entry Display to Selection List

of these fields, and will request the list display (see Figure 5, 4). The user can then make a selection from the list rather than typing the name.

The second type of field on an entry display is a selection field that allows a selection from a fixed set of choices. The choices are numbered as shown in Figure 5, 2 , unless the choice is a value that has significance by itself, as in the case of a command parameter value. The user need only type the number for the desired choice in the same fashion as on a menu. When the prompt requires a Yes or No response, Y and N are accepted for Yes and No, as shown by 3 in Figure 5. In the case of command prompt-entry displays, the actual parameter values are accepted and are listed to the right of the entry field, like *REPLACE, *ADD, or *MERGE.

Command Level Support

While the display interface of the AS/400 system is carefully designed not to require a knowledge of commands, and even to hide commands, most actions result in a command being processed (see Figure 2, bottom). The terminology for the options and choices shown on displays closely matches the terminology for the corresponding spelled-out names of commands and parameters. This, coupled with the availability of a command line on most menus (see Figure 3) and list displays (see Figure 4), makes it very easy for users to begin using the command fast-path approach for frequently requested functions. A user who knows the command can enter it instead of taking menu options. The same entry displays that are presented if that function is selected by number from a menu or list display are available when entering commands. The entry displays can be requested at any point in typing the command by pressing F4. Any parameters already typed are carried over and filled in on the entry displays. Defaults are shown in the entry fields for any parameters not specified.

Online Help Information

Even with a flexible user interface, the time will come when a user does not understand how to use a display or how to get started on a task. Through the AS/400 help facility, supporting information is immediately at hand.

The help structure defined in IBM's CUA combines help on displays with a help index. The AS/400 help facility provides comprehensive display help and advances the help index concept by giving users a search capability.

The AS/400 help facility provides the type of information users need to complete their immediate task, not a long discussion on how the system or a function works. Rather than duplicate printed manuals, the help facility takes advantage of what the computer does best: provide quick access to specific information. The key to quick access is the information-module concept. All help information is in the form of small building blocks, called information modules, that provide specific bits of information. A single information module can be used individually, or linked together with other information modules in different combinations or sequences.

As shown in Figure 6, the help facility makes use of the information-module approach to provide both context-sensitive help (based on cursor position) and a searchable index of help topics. Context-sensitive help is provided for all displays by associating specific help areas on each display with specific information modules. When a user presses a Help key, the help facility displays the information module associated with the area where the cursor is currently positioned. For example, when the cursor is on a specific line or field, field help is provided, as shown by 1 in Figure 6. When the cursor is in other, nonspecific areas of a display, extended help is provided, as shown by 2 . The extended help consists of information modules on the use of the display as a

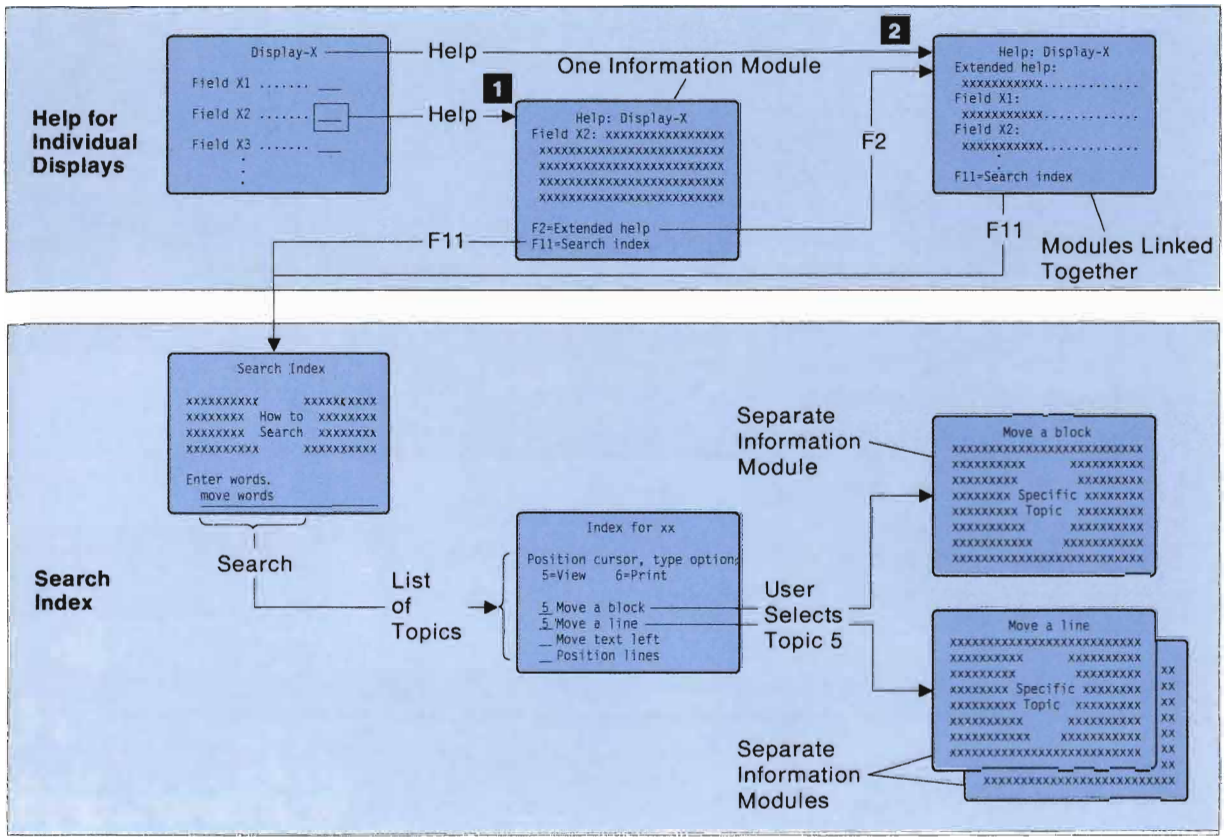


Figure 6 How a User Gets Help

whole, in addition to all of the field help modules describing the use of individual fields. The modules are linked together so that users can move forward and backward through them to see all help for the display. If users initially received help for a specific field, they can get the extended help by pressing a function key (F2).

The help linked to specific displays and fields provides the immediate assistance a user needs to interact successfully with each display. Through the Search Index function, users can get the big picture of how to perform a task that may encompass multiple displays, or, if needed, an explanation of a concept or term they do not understand. Furthermore, the users can ask for the information in their own words, not just the terms used by the system.

Search Index provides a set of online indexes, one for Operating System/400™ (OS/400™) and others for application packages. The index searched is determined by the product being used at the time the search is requested. If AS/400 Office is being used, the Office index is searched. The index consists of a list of topics, each of which is linked to one or more information modules.

As shown in Figure 6, a user can request Search Index from help by pressing a function key (F11). Although the index is used most effectively by entering search words, the user has the option of viewing the entire index by simply not entering words. If the user does enter search words, each of the words (except for words used as simple connectors, such as *the* or *of*) is matched against tables of keywords and synonyms, and a list of the topics that best match the user-entered words is displayed.

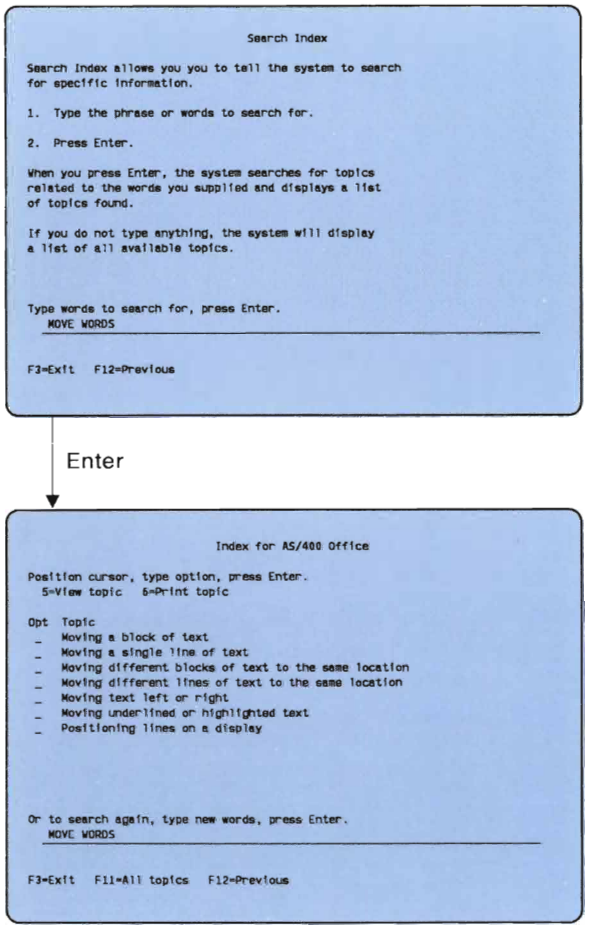
Figure 7 shows an example of a user searching the Office index. The user enters MOVE WORDS. The search process compares MOVE with the keyword and synonym tables and finds matches for MOVING and POSITIONING. Similarly, comparing WORDS with the keyword and synonym tables finds matches for TEXT and LINES. As a result, the user is presented with a list of topics on MOVING TEXT and POSITIONING LINES.

Conclusions

The AS/400 system takes the familiar and proven features of current systems and introduces many state-of-the-art advancements in work station ease of use.

The AS/400 system introduces new dimensions in user interface consistency and offers consistency with the future direction of other IBM SAA systems. Even more importantly, it introduces consistency between attached personal computers and dependent work stations, without compromising the potential and strengths of either.

An improved list display is used to simplify creating and working with objects. It allows actions to be performed directly on the objects in the list or by typing the name. This allows all actions to be performed from within the list area, simplifying and streamlining work.



RSLL365-2

Figure 7 Example of Search Index Displays

Whenever an action is requested that requires additional information, an entry display is provided that layers the request, with frequently used options presented first, and then, on request, more advanced options. Options whose applicability depends on other responses are presented only if appropriate.

At any time users can ask for help and receive text describing the current field. In addition, they can request more information by typing words describing what they want to know. In response, they receive a list of topics that satisfy their request, from which they can choose the ones displayed.

With the introduction of the AS/400 system comes an advanced integrated user interface that spans the comprehensive facilities of this mid-range system. The user interface is designed to be used by a broad spectrum of users, and provides them with interface capabilities not previously available to users of general-purpose computers. The interface is designed to allow each user to grow in productivity, using menus, layered entry displays, list displays, and command lines that are backed by a sophisticated indexed help structure. The interface capabilities can continue to be extended with other methods of interaction as CUA is extended, preserving consistency between CUA-complying products and a state-of-the-art interface.

References

1. Botterill, J.H., *The Design Rationale of the System/38 User Interface,* **IBM Systems Journal,** Volume 21, Number 4, 1982.
2. **Systems Application Architecture, Common User Access: Panel Design and User Interaction,** SC26-4351, December, 1987.

An Integrated Data Base

Describes the AS/400 integrated data base that can appear as multiple, interface-specific data bases using a single data base manager and storage representation.

Mark J. Anderson and Richard L. Cole

Introduction

The AS/400™ data base is different from traditional system data bases because of its innovative design which integrates the data base with the operating system software and integrates support for several different interfaces into a single data base manager. The design goal for the AS/400 data base manager was to support application migration from the System/36 and System/38, provide Systems Application Architecture™ (SAA™) support, and allow for future data base enhancements. Therefore, the disk data management interface of the System/36 and the data base interface of the System/38 are supported as an integral part of the AS/400 data base. Also, the interactive data definition utility (IDDU) and the SAA data base interface, called structured query language (SQL), provides data dictionary and relational data base interfaces to the AS/400 data base. The single, generalized data base manager understands all functions necessary for these interfaces, ensures they conform to their definitions, and coordinates their interaction.

Choosing an Integrated Data Base

An integrated data base design allows applications written for each interface to coexist and operate using the same data. Because a single data base system manages a single storage representation of data, users may choose the interface appropriate to the application they are building. For example, an application containing embedded SQL can be used to do queries or mass updates, while another application using the more efficient AS/400 data base techniques could be used to randomly update records. Also, programmers with a System/38 background can use the Operating System/400™ (OS/400™) System/38 environment support to create and manage data base files, while other users could use the simpler IDDU interface.

The traditional approach to satisfy these diverse requirements is to build separate, independent data base data management systems for each interface. This approach requires data to be replicated in each data base. Besides the obvious disadvantage of outdated or inconsistent data between the separate data bases, users are burdened with extracting data from one data base and moving it into another. Even when a single representation of data is maintained, the traditional system has separate, nonintegrated data base managers. These multiple data base managers are not coordinated and, between them, lack data integrity, concurrency, and usability.

An integrated data base is easier to manage. Support for a single data base means that saving data, journaling data base changes, recovering data, controlling data authorizations, and so on, are much easier because only one set of system functions or commands must be learned. Other implementations require users to learn new data base object management procedures for each data base product.

Also, a single data base avoids redundant control requirements. For example, with nonintegrated data base managers, users denied access to data in one interface might obtain it through another, unless all independent interfaces had been similarly restricted. Any function used by a particular interface that has persistent operational ramifications, or any constraint (such as an index that enforces the uniqueness of key values) that is applied through one interface, is enforced through all interfaces. For example, once a file is journaled, all changes to the file's data are journaled, regardless of which interface started journaling or which interface is changing the data.

Additionally, an integrated data base provides users with a much easier way to migrate from one interface to another. For example, a System/36 user that wishes to modify an RPG II application to take advantage of the additional capabilities of embedded SQL can do so one program at a time. The converted program can be used concurrently with any of the unconverted programs. If the traditional approach had been chosen, the complete set of programs and all files would need to be converted at the same time, because a migrated application either has a separate data base or an incompatible data base manager. Having to convert all an application's programs and files at the same time makes it impractical to use new functions in current applications.

In addition to improving end-user productivity, the integrated data base approach allows for more productive systems software development. The

integrated approach implements any given function only once instead of once for every interface. Because the functions of each interface overlap one another, the amount of logic required to develop and maintain the system's data base support is decreased.

AS/400 Data Base Data Management

AS/400 data base data management has facilities to define and manipulate data, process queries, maintain file cross-referencing, record data base changes, and manage transactions. These facilities are part of OS/400 and are general enough to provide an integrated mechanism for data storage and access.

The data base is made up of two types of files: physical and logical. Physical files contain the actual data and may be considered tables, having records for rows and fields for columns. All records in a physical file have the same field attributes and record length. A logical file provides alternative definitions of data to support application-data independence and to avoid redundancy. Logical files allow users to see records in different sequences, select subsets of records from physical and logical files, map fields to different data types, reorder fields, and select subsets of fields.

The four categories of logical files are:

Simple logical files that map data from a single physical file to another logical record definition.

Join logical files that define a single record definition built from fields of multiple physical files.

Multiple format logical files that allow access to several physical files, each with its own record format definition.

View logical files that are created by the SQL CREATE VIEW statement and define a single record definition built from fields of multiple physical and other view logical files.

Records are stored in physical files in the sequence that they were added (arrival sequence). All file types can access records in arrival sequence. Physical files and simple, join, and multiple format logical files can use indexes to access records in logical sequences based on the contents of data fields (keyed sequence). Fields controlling the logical sequence of records are called key fields.

The data description of a file can be contained within the programs that use the file, can be part of the file itself, and can be in an IDDU data dictionary. If the file is described only by the programs that use it, it is called a program-described file. Program-described files cannot be processed by programs like the Query utilities that rely on the data base to provide field definitions.

Several methods can be used to describe files to the AS/400 data base. One of these is data description specifications (DDS). Using DDS, users can describe all types of data base files except view logical files. These definitions are then used to create the file. View logical files and physical files can be created using SQL CREATE statements. A file created with DDS or SQL is called an externally described file and has its field definitions stored with it. Externally described files can be accessed by utilities like Query and they permit programmers to simplify programs by leaving out data definitions the files can supply. (For more information, see the article *Application Development Support*.)

Another way of describing files to the AS/400 data base is to use IDDU. Files created from IDDU definitions are called dictionary-described files. Like externally described files, dictionary-described files can be used by programs that require files with field definitions. If a program-described file already exists, IDDU can link a definition to it. This makes the file dictionary like the Query described, and usable by program utilities. If an externally described file already exists, IDDU can also incorporate that file's definition.

Powerful data manipulation functions are provided by the data base. Non-keyed files can be processed sequentially by arrival sequence or randomly by relative record number. Keyed files can be processed sequentially by arrival or keyed sequence, or randomly by relative record number or key value. Records can be added, deleted, or updated. A file can be cleared of data, physically reorganized using a specified physical or logical file's keyed sequence (reorganization may also remove deleted records from the file), or initialized with a set of deleted or default record images.

Query processing allows relational selecting, projecting, joining, grouping, and ordering of records. The integrated support provides a single source for query validation, optimization, and implementation of SQL statement processing, several end-user Query utilities (including those utilities running in the System/36 and System/38 environments), and other system functions. This level of integration is possible because the system interface to the query support is independent of any user interface, and is functionally capable of supporting all user interfaces. Each user query is compiled or interpreted into this system interface.

Data base files are managed with generic functions that rename, move, change the attributes of, authorize, and save them. These functions are generic in the sense that, not only the data base files, but all user objects on the AS/400 system, are managed using the same commands and utilities. A single set of generic functions can exist because the AS/400 data base manager is part of the AS/400 system. These generic functions can be used by any data base

interface and changes made in one take affect in all.

The AS/400 data base manager maintains a set of files that contain basic attribute and cross-reference information about all files. These files, like any other data base file, can be queried by users. The cross-reference information tells which data dictionary describes each dictionary-described file, how files are interrelated, and how they are dependent on each other. The data base manager uses this cross-reference information to build catalogs for the SQL interface and to generate reports on where data definitions are used for IDDU.

Journaling records changes that users make to data. Users may use the journal to: help recover if a file is damaged; decrease the time required to save; provide an audit trail or activity report; and provide job accounting information. The AS/400 data base can treat multiple changes to a file's data as a single transaction. At the end of the transaction, the changes can be committed or rolled back. When the system or job ends abnormally, any uncommitted changes are automatically rolled back.

Data Base Interfaces

AS/400 data base uses a single operating system-level representation for all files and an integrated set of functions to operate on those files. The files and functions are available to any interfaces that can support them. Figure 1 shows this structure of interfaces to the AS/400 data base.

While all facilities are available to all interfaces, some use a subset of them. For example, because the syntax of the SQL language cannot describe multiple format files (files composed of more than a single record definition), such files are not allowed in libraries created to hold SQL files. The AS/400 data base manager can ensure the consistency of the SQL interface because it knows the file types that are not allowed and the libraries created primarily to hold SQL files.

System/36 Disk Data Management Interface

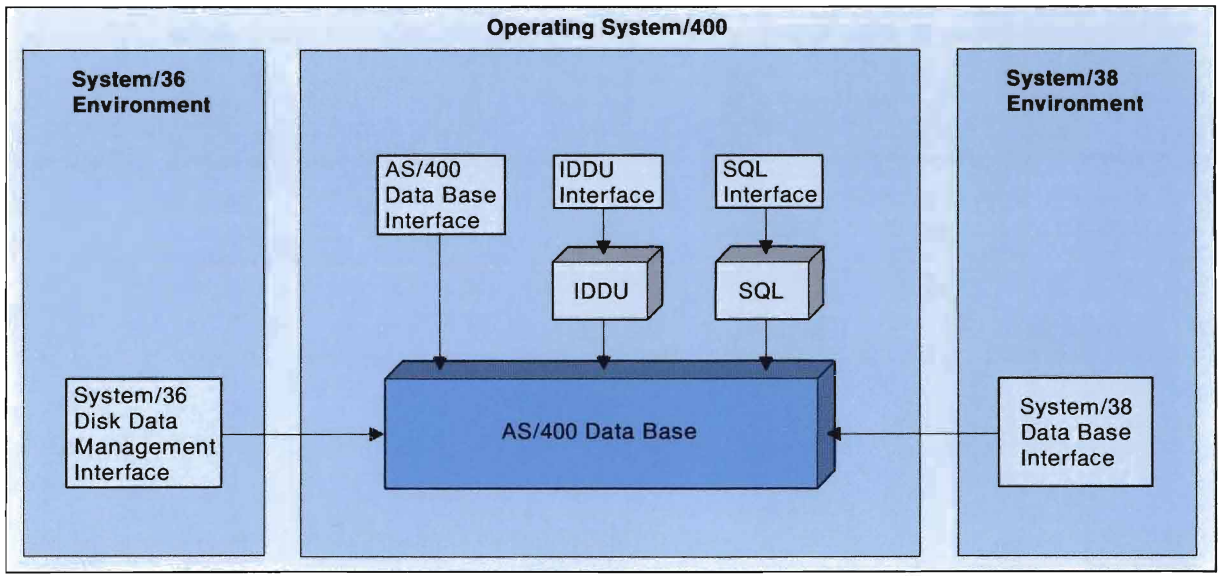
System/36 disk data management supports four basic file types: sequential, keyed, direct, and alternative index. Sequential and direct files are implemented as simple non-keyed AS/400 physical files having no field-level definition. System/36 keyed files are implemented as keyed AS/400 physical files that contain fields as required to represent the key definitions. Alternative index files are simple logical files.

Because of the integrated data base, users of the System/36 environment can access files created using other interfaces. A System/36 application that can migrate need not know whether the file was created from within the System/36 environment or System/38 environment, using SQL or IDDU. This level of transparency in the interface is possible because the AS/400 data base manager understands all types of data base files; the System/36 file support is an integral part of the data base manager. (For more information about the System/36 environment, see the article *The System/36 Environment*.)

System/38 Data Base Data Management Interface

The System/38 data base interface uses all of the AS/400 file types except view logical files. The data definition, data manipulation, query processing, generic object functions, file journaling, and commitment control used by the System/38 interface are all subsets of the AS/400 support.

Again, because the System/38 file support is part of the integrated data base manager, the



RSLL394-4

Figure 1 Interface to AS/400 Data Base

System/38 data base is not restricted to files created using the System/38 data base interface. All files may be processed regardless of the interface used to create the file.

Interactive Data Definition Utility Interface
IDDU is an enhanced version of the System/36 data dictionary utility and is a central repository for data definitions. Field, record format, and file definitions can be created and managed independently, and files can be created from these stored definitions.

The IDDU data dictionary can be used in a passive or an active mode. In passive mode, users must keep data definitions synchronized with the files they describe. The IDDU support provided on the System/36 was primarily passive. Users could unlink, change, and relink a file's definition without restructuring its data.

The AS/400 IDDU data dictionaries are used in an active mode, where the system keeps the definitions synchronized with the files they describe. If a file is created using IDDU data definitions or an externally described file is described by a data dictionary (created through some other interface such as DDS or SQL CREATE statements), the AS/400 data base manager automatically reflects to the data dictionary changes made to the file. The data base manager knows that a file is linked to a data dictionary, and therefore can maintain consistency between them. Users are prevented from modifying definitions while the definitions represent externally described files.

IDDU data dictionaries are made up of a set of related data base files that contain the definitions. Therefore, users can query the data definitions in a dictionary or access them from a program. However, the data base files containing the data dictionary are protected from direct changes by users.

Files created by IDDU are externally described, and so have a copy of their definitions stored with them. Therefore, accessing the data dictionary is not necessary for data base operations, and dictionary contention problems are avoided. The files are then portable, allowing another AS/400 system without a data dictionary to use them.

Structured Query Language Interface
AS/400 Structured Query Language/400 provides the IBM SAA data base interface and is an implementation of the relational data model that describes operations on tables, rows, and columns. SQL supports powerful data definition and data manipulation statements. For example, the SQL CREATE VIEW statement can create an alternative view of a sales representatives' salary table that presents average salaries by department. Or, a single SQL UPDATE statement can add 10% to the salaries of sales representatives who exceed their quota by 50%. On the System/38 or System/36, the same functions require program logic.

SQL statements can be issued interactively or embedded in application programs. Depending on the type of statement, either type of use results in a call to the data base query support to run it or generate an intermediate representation of the query statement for storage with the program for later processing.

SQL-created tables are implemented as non-keyed physical files, SQL views are view logical files, and SQL indexes are simple keyed logical files. The implementation of SQL views is particularly interesting because it combines a data base file with AS/400 Query. When an SQL view is queried or opened using any interface, the data base manager calls query processing to perform the relational functions defined for the view.

Files created using SQL can only exist in special libraries created using the SQL CREATE DATABASE statement. These SQL libraries contain a journal, a journal receiver, an IDDU data dictionary, and logical files, constituting a catalog that describes SQL-created files. The AS/400 data base manager prevents any file that cannot be described by the catalog (such as program-described files and certain logical files) or that cannot be processed by SQL (such as multiple format logical files) from being created, restored, or moved into a library created using SQL. This ensures the catalog contains only relational files and that it completely describes the files in the data base.

Any table created using SQL is automatically journaled so commitment control can be used. Any file created into an SQL library (using SQL or any other interface), moved into an SQL library, or restored into an SQL library is automatically described by the catalog because the AS/400 data base manager incorporates the file's definitions into the data dictionary.

Files in SQL libraries can be accessed using other interfaces. Also, SQL statements can be used on files in libraries not created to hold SQL files, thereby allowing access of files created using any other interface.

Conclusions
The AS/400 system's innovative approach to data base system integration offers flexibility for:

- Meeting today's requirements of compatibility and coexistence with existing systems.
- Improving data definition and query facilities using IDDU and SQL.
- Staging conversion of existing applications to incorporate enhanced functions.
- Meeting future application requirements by allowing data created using one interface to be accessed by another.

Future applications require support for increased transaction rates, very large data bases, distributed data base management, and so on. Support and management of these and other features is simplified and enabled as a result of the AS/400 integrated data base manager.

Acknowledgments
The authors wish to thank William S. Davidson and Alvin G. Grossbach for their contribution to the content of this article.

Application Development Support

Describes the features of the AS/400 application development support, which allow productive application development.

Gary R. Karasiuk

Introduction

The AS/400™ system contains an advanced set of tools and system functions that enable users to productively develop applications. Highlights include integrated data base functions, the source editor, compilers and the debugger, and work station support. Specifically, productivity is improved by:

- Using externally described data to reduce redundant data descriptions and pass information across the different phases of application development. In our model of the application development life cycle, the phases are: requirements, analysis and design, produce, build and test, and release and control. [1]

- Integrating the source editor with the system (and especially the compilers) to improve the productivity of the produce phase.

- Integrating the debugger with the system to improve the productivity of the test phase.

- Accommodating users with different system backgrounds (System/36 and System/38).

Shared Data Descriptions

One of the important software development trends of the 1980's is the simplification of application programming by moving some of the code from procedural programs into a declarative form, which is usually a part of the data model description. This results in two clear advantages: reusability and simplicity. One example of reusability is moving data descriptions out of the program and into the data model. This allows users to have a single authoritative source for their data descriptions. This simplifies maintenance, as users can be assured they are looking at the correct description, and if they choose to change the description, they are changing the only occurrence of the description. This improves the quality of the application, because different data descriptions do not exist for the same piece of data, and also reduces the amount of coding. In a conventional system, if a user had an application with 10 programs that accessed the same file, and wished to add a validity check to one of the fields, the user would have to add additional logic to each of the 10 programs. If the validity check could be added to the data model, it would only have to be added once, as is the case with the AS/400 file model and its use of externally described data. Device files on the AS/400 system contain externally described data, which is stored with the file when it is created.

All device files can be described at a **field level**, with field attributes such as data type and length. Options exist that allow specification of such things as descriptive text and field validation parameters for each field. The normal unit of data transferred by a program is a **record,** which is made up of one or more associated fields. This collection of fields is called a **record format.** This information is entered only once and then is used by other components in the system (see Figure 1).

Many of the devices on the AS/400 system support the common file model, and thus allow input/output (I/O) redirection. Some of the different device file types are physical (for storing data); logical (different views of physical data); display (for displays); printer (for page formats); and communications (for data exchange). Applications, for the most part, can be written so that they are unaware of the underlying device file type. Consequently, files can be overridden at run time. One novel use of this capability is to replace display files with communications files, to provide for automatic testing of interactive applications.

A typical application development scenario illustrates how data descriptions are passed in the AS/400 system from the design phase to the produce phase:

1. The internal data definitions needed by the application are entered into a reference file.

2. The screen design aid (SDA), a utility for designing displays, is used to define application displays. The fields to appear on the displays can have their definitions included from the reference file, to ensure consistent definitions of the same data. Integrity information stored with the reference fields ensures that integrity checks are performed when the display file is accessed. Also, displays can be quickly strung together to form a simple prototype of the application, allowing for early end-user feedback.

3. The physical and logical files are created, again with some of the field definitions from the reference file.

4. The compilers (RPG, COBOL, command language (CL), PL/I, and BASIC) have language extensions to extract data definitions from files and convert (back translate) them to high-level language data structures. (See Figure 2 for an example of back translation.) Compiler directives specify which record formats are to be back-translated.

5. Other utilities also make use of the externally described data. The data file utility (DFU) creates applications that add, delete, and update data records. AS/400 Query creates reports that include features such as breakline processing, sorting, and summation. Using the externally described data, Query, for example, could extract the column headings that appear on the final reports from the files. Both DFU- and Query-generated programs are used to supplement the other high-level language programs in the application (typically RPG or COBOL).

The AS/400 file model also makes it easier for the application developer to take advantage of system function, and thereby save application code. This is a natural result of the clean interface between application programs and files. The savings in application logic (code) is shown in these two examples:

Subfile Support: The logic of scrolling lists of items on the work station can be handled by the system through subfile support. It is the system that processes the positioning of the list. The application need only define the characteristics of the subfile, such as the maximum number of items in the list and the format of a line in the list.

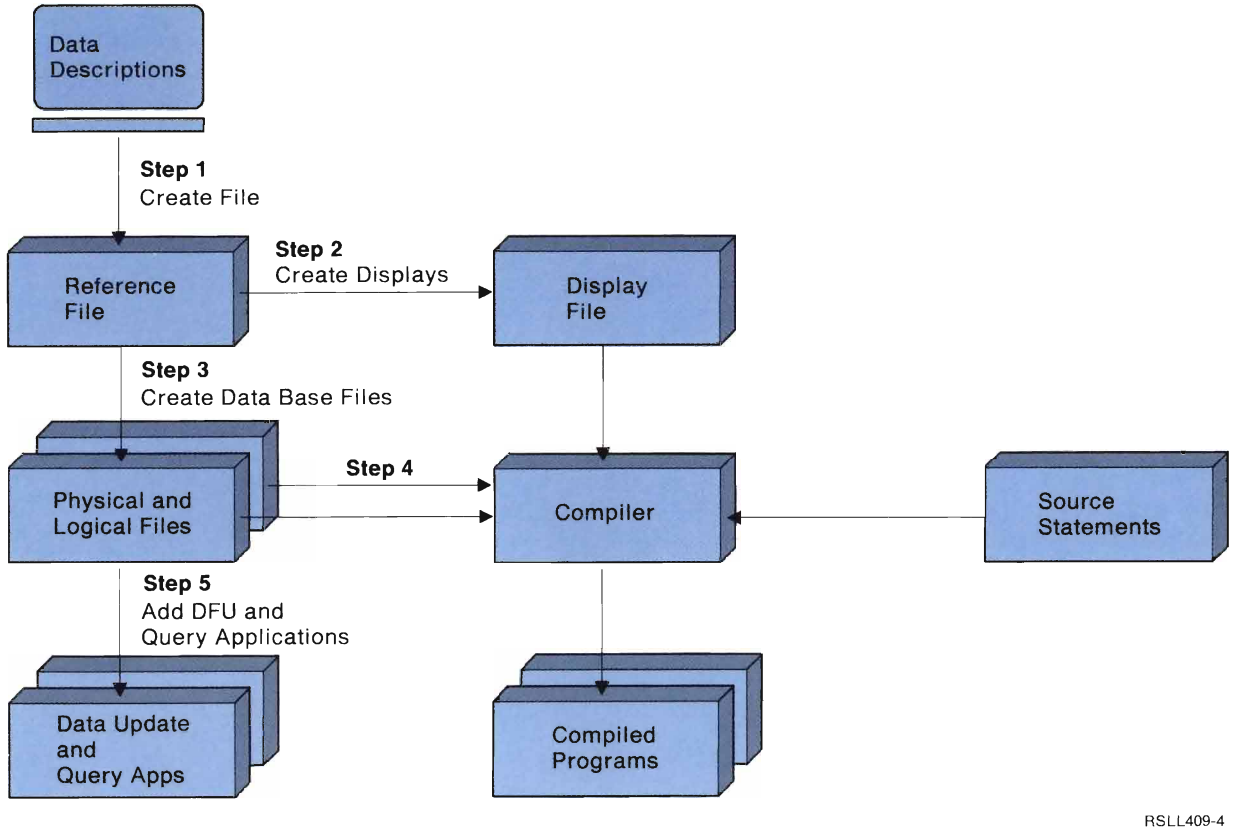


Figure 1 Shared Data Descriptions

Select/Omit Support: The logic of identifying a subset of data base records can be handled by the system through data base select/omit support. This saves the applications from reading all of the records in a file and performing their own selection logic.

The System Editor

The AS/400 system has chosen to concentrate on the produce phase of the application development model. As a result, the editor is well-integrated into the system. The editor on the AS/400 system is the source entry utility (SEU).

The editor is integrated with the compilers with its syntax-checking support. All of the languages on the system support interactive syntax checking. The editor is responsible for determining the syntactic boundaries of the statement, and the compiler is responsible for the actual syntax checking of the statement. Splitting up the function in this manner has three advantages. First, syntax checking is consistent from the user's perspective, because only the editor is responsible for the end-user interface. The options that control syntax checking are the same for all languages. Second, the compiler syntax checkers and the interactive syntax checkers are

The following is an example of a payroll record description in DDS.

```
A          R PAYREC                    TEXT('ORDER RECORD')
A            NAME          50A         TEXT('Full Name')
A            ADDRESS       100A        TEXT('Street Address')
A                          50A         COLHDG('Street' 'Address')
A            SALARY        8  2        TEXT('Annual Salary') EDTCDE(1 *)
A                                      COLHDG('Annual' 'Salary')
A            DEDUCT        7P 2        EDTWRD('$    0.  &CR')
A                                      TEXT('Deductions')
```

This is the way the description would be automatically expanded by the PL/I compiler.

```
DCL 1 PAYROLL-RECORD,
%INCLUDE PAYROLL(PAYREC,RECORD,PR-);
/* ---------------------------------------------------------------------- */
/* PHYSICAL FILE: PAYROLL.KARSLIB                                         */
/* FILE CREATION DATE: 87/10/13                                           */
/* RECORD FORMAT: PAYREC                                                  */
/* RECORD FORMAT SEQUENCE ID: 37B899FF85E16                               */
/* -------ORDER RECORD---------------------------------------------------- */
15 PR-NAME       CHAR(50),          /* Full Name                          */
15 PR-ADDRESS    CHAR(100),         /* Street Address                     */
15 PR-SALARY     CHAR(8, 2),        /* Annual Salary                      */
15 PR-DEDUCT     DEC(7,2);          /* Deductions                         */
```

RSLL408-1

Figure 2 Sample Back Translation

less likely to diverge because both products are developed together. In fact, in some cases, the syntax checking is performed by the early phases in the compiler. And, finally, the structure of the editor is simpler.

The main limitation of this approach is that not all syntax (and few semantic) errors can be detected due to the statement-by-statement nature of the syntax checking. Also, due to the richness of some of the languages, it is difficult to quickly determine the statement boundaries. In the case of PL/I, heuristic methods were needed to determine the statement boundaries.

The editor also supports prompts for different languages. The language prompt support varies by type of language. For CL, the system prompt function is called from inside the editor to provide very complete statement prompts. The prompt function provides command formatting, layered prompts, and context-sensitive help text. (See the article *An Integrated User Interface* for more information on the system prompter.) For RPG, the prompt function provides field-level support for each of the RPG specifications, again with context-sensitive help; for BASIC, the prompt function calls the BASIC session manager. Other languages have simpler prompt support.

The editor also supports a split-screen mode, where a compiler listing can be browsed on one half of the display while corrections are being made to the corresponding source member on the other half.

Debugging

The AS/400 system has an integrated symbolic debugger that is built into the microcode and into Operating System/400™ (OS/400™). One of the unique characteristics of the AS/400 debugger (such as, setting and stopping at breakpoints) is that only a small performance penalty is paid when using it. The debugger's high-level performance is achieved through the event mechanism that is built into the system microcode. (An event is signaled by the system each time an instruction is processed so that OS/400 can monitor the progress of the code being run.) Programs that are compiled with the debugging option turned on (which is the default) have debugging tables that are generated along with the object code. (These debugging tables do occupy additional storage.) Users that always generate the debugging tables as a part of the compile step have the flexibility to debug any program in their application without having to recompile.

The debugger supports all of the common debugging functions:

- Set breakpoints at high-level language statement numbers
- Display and change high-level language variables
- Issue any command while stopped at a breakpoint
- Trace

The debugging support allows users to closely monitor the application's processing. It also increases the value of a testing session, as the user can temporarily correct many types of errors (by changing variables and issuing system commands), allowing other errors to be found before the program must be recompiled.

Through group job support, the user can have the editor running in one group job and a debugging session in another. The user can hot key between the jobs as the need arises, allowing simpler errors to be corrected while debugging.

Another system feature that aids in the debugging phase is dynamic binding. The design of the AS/400 system makes the link-edit step unnecessary, allowing the user to compile and run. Corrections can be compiled into test libraries, which, for the programmer, are placed ahead of the production libraries. Program calls are resolved at processing time, causing the test versions to be called. This allows programmers to test corrections without having to have a separate test version of the entire application. At the same time, others can continue to use the production level of the application.

Accommodating Different User Sets

A unique requirement for the AS/400 system was the need to accommodate users from the System/36 and the System/38. The application development support enhances the functional richness and ease of use of these two systems by providing consistency, flexibility, and the ability to migrate easily.

End-user interface changes were made to all of the application development support. The goal was to make all of the products more consistent and, thus, easier to use. Another goal was to make the application development support more consistent across IBM's entire product line, including Multiple Virtual Storage (MVS), Virtual Machine (VM), and Operating System/2™ (OS/2™). With these changes came additional ease-of-use features, such as more online help information and better field prompts.

Additionally, calling the application development support has been made more flexible. Users now have three ways of calling this support: through the programming development manager (a utility that presents the user's programming objects in list form), the command shell, or the Programmer's Menu. The programming development manager allows users to look at their programming objects (files, programs, and the like) and then select the appropriate function. Many of the functions have been generalized, such as the compile function, which can be applied to any source type. The programming development manager also allows users to create their own user-defined functions that can then easily be applied in any of the programming development manager lists. So, users can apply functions to objects (through the programming development manager), objects to functions (through the Programmer's Menu), or both at the same time (through commands).

To accommodate the wide difference in user-experience levels, some products have expert modes, which remove some of the help information from the display to make more room for the user's data. For example, the programming development manager allows the user to turn off displaying the options and the command keys to allow more room for the object list, thereby showing 17 items on the display instead of eight. And, some products support a fast path. The DFU fast path bypasses many of the normal DFU prompts (the system picks appropriate defaults), allowing simple DFU applications to be generated quickly.

And, finally, to make the migration from System/38 and System/36 easier, multiple dialects of the various languages were added. For most applications, this enables System/38 and System/36 users to recompile their existing programs unchanged. This has also been extended to the screen specification languages, where both the screen format generator (SFGR on System/36) and DDS (System/38) are supported.

AS/400 utilities also support these additional languages. Screen design aid supports creating SFGR source as well as DDS source. DFU runs applications that were created using System/36 DFU.

The AS/400 Migration Aids were developed to facilitate moving applications, system data, and user data from the System/36 and System/38 to the AS/400 system. This product allows some migration work to be done ahead of time by providing an extensive set of facilities that run on the System/36 or the System/38. These facilities include:

- Analysis reports, which will find the programs that require change to recompile successfully. Individual source statements are flagged with either warning or error messages, allowing the user to identify the problem areas.
- System reports, which show the amount of data to be migrated. They also show what has and has not been migrated or analyzed. This allows the migration to be performed in stages.
- A facility for automatically determining the source type (such as RPG, COBOL, Query) for source members that did not have a source type previously specified.
- A facility for reconstructing source from compiled menu, message, and SFGR objects.

Conclusions

The thrust of the AS/400 application development support is to provide an integrated set of tools that focus on the produce phase of the development model. The more complex applications of the future will need advanced tools to achieve the necessary productivity and quality. The future will see more emphasis placed on the other phases, especially the analysis and design and build and test phases.

A secondary thrust of the application development support is to move function out of procedural programs and into declarative forms, and also to take advantage of system function. This is an initial step toward controlling the complexity of application programs. The application development support provides a base set of function that meets the challenges of today and is expandable to meet the challenges of the future.

Acknowledgements

The author would like to thank many of the people in the Toronto Programming Center for their help with this article, and especially Steven J. Hodson and Claus Weiss.

References

1. Hoffnagle, G.F. and W.E. Beregi, *Automating the Software Development Process*, **IBM Systems Journal**, Volume 24, Number 2, 102-105, 1985.

The System/36 Environment

Describes an environment which supports System/36 applications and users on the AS/400 system.

John A. Modry, Peter J. Heyrman, and Steven A. Dahl

Introduction

The System/36 environment is a feature of the AS/400™ system that supports developing and running System/36 applications that use procedures, Operation Control Language (OCL) statements, utilities, menus, messages, and application program interfaces (APIs). Most System/36 applications can be easily migrated to the System/36 environment and just as easily migrated back to System/36. A number of challenges were faced in providing equivalent function and interfaces with a preceding system while still taking advantage of the new function and improved usability of the AS/400 system.

The primary design goal of the System/36 environment was to allow System/36 applications to work on the AS/400 system, both functionally and in terms of the interfaces seen by the users of the applications, without requiring source code changes. The System/36 environment is operating system support that is designed to provide System/36-equivalent function, using underlying AS/400 facilities and constructs wherever possible. Any AS/400 function can access, update, delete, and rename the migrated objects from a System/36. The user's compiled programs, messages, display formats, data file utility (DFU) programs, files, libraries, and so forth, are all AS/400 programs or objects when being accessed or run by the system. Two significant advantages of the System/36 environment approach are:

- The performance of the System/36 environment is approximately the same as using equivalent Operating System/400™ (OS/400™) function.
- The System/36 environment provides access to AS/400 functions. This allows most System/36 applications to be expanded to use AS/400 commands and programs without requiring that the application programs be rewritten, and allows interactive users to call AS/400 commands and programs while in the System/36 environment.

A variety of approaches were used to design the System/36 environment. For some functions, the complete System/36 design was used and re-implemented for the AS/400 system. For other functions, extensions were incorporated at various points within the AS/400 system to provide the functional equivalent of System/36 support. In some cases, System/36 user interfaces were extended to be more functional and to achieve consistency across the entire AS/400 system. The different approaches blend together to produce a System/36 environment on the AS/400 system that provides support for System/36 applications, while maximizing performance, usability, and extendibility.

Structure

The System/36 environment consists of AS/400 objects that represent various parts of a System/36 application, as well as the programs, objects, and the like that support the System/36 environment.

Object Structure

On System/36, applications are stored in files and libraries. Four types of library members exist:

- Source members contain editable information that is input to another process, such as a compilation. Examples of source members are high-level language source statements, message source, and display format source.
- Procedure members contain OCL statements that are similar in function to control language (CL) statements on the AS/400 system. System/36 procedures are interpreted by the System/36 Reader/Interpreter.
- Load members are the internal form for objects, such as compiled programs, display formats, message members, and configurations.
- Subroutine members are the output from a process such as compilers, Query, or DFU. On System/36, program subroutines are combined to create load members.

Figure 1 maps some key System/36 objects to AS/400 objects.

On the AS/400 system, source and procedure members are mapped to source files so a single editor can change source statements, procedures, CL program source statements, and so forth. This eliminates the need to learn multiple editors to change source members.

System/36 Object	AS/400 Object
Library	Library
Source Member	Member of Source File QS36SRC
Procedure Member	Member of Source File QS36PRC
Compiled Program (RPG II and COBOL)	Program
Subroutine Member	Program
Compiled Display File	Display File
Compiled Message Member	Message File
File (Sequential, Direct, and Indexed)	Physical File in Library QS36F
Alternative Index	Logical File in Library QS36F
Virtual Disk	Shared Folder in Library QDOC
Folder	Folder in Library QDOC
Documents	Documents in Library QDOC
Data Dictionary	Data Dictionary and a Set of Files Within a Library
Library #LIBRARY	Libraries #LIBRARY and QSSP

RSLL397-3

Figure 1 Mapping of System/36 to AS/400 Objects

Support has been provided to handle the special System/36 attributes for an object. For example, the System/36 procedure attributes (such as multiple requesting terminal (MRT) indicator, and log statement indicator) and System/36 source attributes (such as never-ending-program (NEP) indicator, and maximum number of MRT requestors) are supported. In addition to the existing System/36 attributes, new attributes were defined. One of these attributes indicates that a program was compiled for the System/36 environment and must be run in the System/36 environment.

User profiles on the AS/400 system support all of the System/36 user profile attributes, including: initial sign-on menu, initial sign-on procedure or program, initial current library, mandatory-menu attribute, and mandatory procedure or program attribute. In addition, a new user profile attribute indicates if a user's job should have access to System/36 environment functions.

The library structure for the System/36 environment has been changed from the library structure of System/36. On System/36, the system library, #LIBRARY, contains all of the IBM-supplied objects for the System/36 operating system (System Support Program, or SSP). Because #LIBRARY is always checked when searching for objects, customers often place applications that are used by many users in #LIBRARY. On the AS/400 system, library QSSP contains all of the IBM-supplied programs, procedures, and files for the System/36 environment, and #LIBRARY is used to hold user applications. This two-library approach allows new operating system releases to be installed without affecting the applications in #LIBRARY.

To maintain information about the System/36 environment, a System/36 definition object has been created (object type *S36). This object includes information about:

- Display stations, printers, the diskette unit, and tape units to be used in the System/36 environment.

  The System/36 environment maps the AS/400 10-character device names to two-character names. This allows System/36 applications that use the two-character device names to be migrated to the System/36 environment.

- System/36 environment file information.

  The default library that contains files is QS36F. This library name can be changed with the System/36 environment definition support.

- Session information.

  This includes information on items such as the default library for a display station, the printer associated with a display station, and so on.

- Spool information.

  This includes printer lines per page, characters per inch, default forms ID, and so on.

- MRT security information.

  This information defines how the System/36 environment controls access to resources used by an MRT.

Tailoring of the System/36 Environment

Tailoring the System/36 environment is an extension of the AS/400 configuration process. Because the System/36 environment assumes

default values for all of the System/36 environment definitions, this tailoring process is optional. The user can tailor the System/36 environment to meet specific needs using the Change System/36 Environment command.

On the AS/400 system, all jobs operate in a subsystem (the AS/400 concept of a subsystem should not be confused with the System/36 concept of an Interactive Communications Feature (SSP-ICF) subsystem). Jobs running in a subsystem can be controlled independently of jobs in other subsystems. The System/36 environment support is an element of the AS/400 subsystem support. The System/36 environment sets up the necessary control blocks that allow the System/36 environment to maintain information about the current subsystem and the System/36 environment. This includes a lists of procedures, MRTs, and other subsystem information.

When starting a subsystem, the System/36 environment determines if a System/36 environment definition object exists. If the object does not exist, the System/36 environment automatically creates the object and supplies default values for all of the definition information. These defaults include defining System/36 environment display stations, printers, the diskette unit, and tape units based on the AS/400 hardware configuration.

In addition to automatically creating the System/36 environment definition object, hardware devices are automatically added to the System/36 environment definition object when they are added to the AS/400 system. When a display station, printer, diskette unit, or tape unit is added or removed from the system, the AS/400 configuration support notifies the System/36 environment, which changes the System/36 environment information for that device. This combination of the AS/400 support and System/36 environment support allows customers to attach a new input/output (I/O) device and immediately start using it.

In addition to the definition process of the System/36 environment, customers have the opportunity at initial program load (IPL) to change certain system values to tailor their system. One of the system values determines if all user profiles should be given access to the System/36 environment. This allows the system administrator to set a single value instead of setting the System/36 environment attribute in multiple user profiles. The System/36 environment system value can be overridden by the individual user profile. Another system value specifies whether the system should create device names in the two-character System/36 format or the 10-character AS/400 format.

Accessing the System/36 Environment

These three ways are used to access System/36 environment functions:

- For users who always want to operate in the System/36 environment, a special environment attribute can be set in their user profiles to indicate that the user is a System/36 environment user. When a System/36 environment user signs on the system, the user has access to all functions available in the System/36 environment. Similarly, if a System/36 environment user submits a batch job, the batch job has access to all System/36 environment functions.

- For users who occasionally need access to the System/36 environment, two commands (Start System/36 Environment and End System/36 Environment) are provided that allow a user to enter and return from the System/36 environment.

- For users who occasionally need to run a single System/36 environment procedure, a command (Start System/36 Environment Procedure) is provided that accesses the System/36 environment, runs the procedure, and automatically returns users to their previous environment.

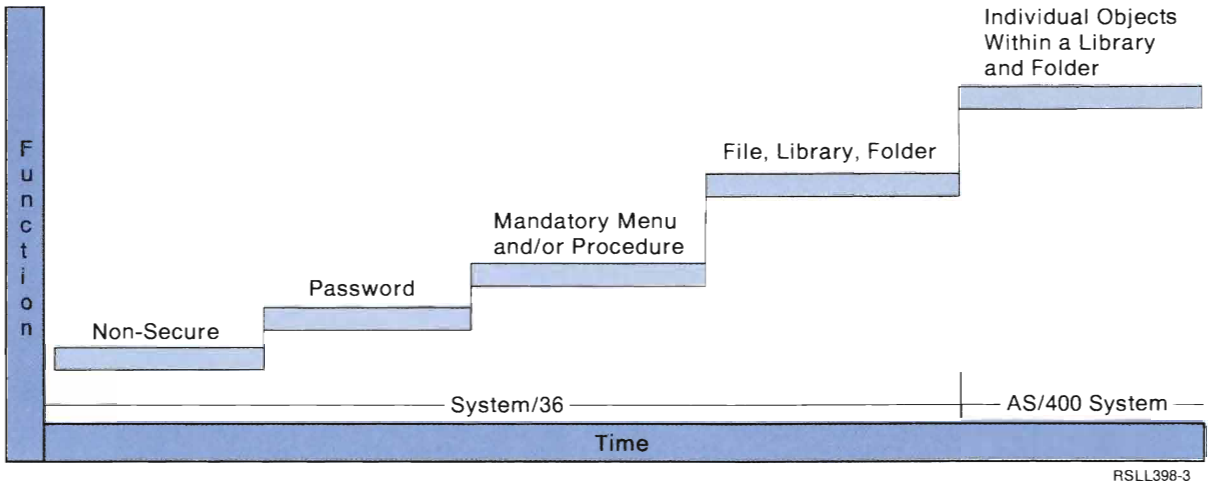


Figure 2 AS/400 Security Time Line

Security

To accommodate both the broad range of security requirements and the differing levels of sophistication of a given customer, the System/36 implemented security so that users can progress through the levels of security as their requirements change. As shown in Figure 2, the user can begin with no security and, as requirements change, progress to requiring passwords and menu security, and eventually to secured libraries and files.

This same philosophy has been adopted and expanded on the AS/400 system. The security administrator can use AS/400 authorization lists to grant and revoke security levels to libraries or objects within a library for groups of individuals. Different levels of security can be selected by the security administrator (unsecured, password and menu, full object-level). The user can grant rights to an individual user or to a group of users for a set of objects. The administrator can, in turn, selectively exclude members of a group from a given resource. The System/36 support for securing a data base file when it is created is provided on the AS/400 system using authority holders. (For a broader discussion on how System/36 security capabilities have been incorporated into the AS/400 security architecture, see the article *Security*.)

User Interface

The user interface consists of the way the user accesses a function and the information seen while the function is running. The System/36 environment supports the System/36 interface used to access a function. The same commands, messages, and other interfaces to System/36 are supported. System/36 users do not have to be retrained to run System/36 environment functions.

Because many System/36 environment functions call AS/400 support, a single interface can be used to manage the system. For example, the AS/400 display to manage currently running jobs is the same as the System/36 environment display. (For more information on the AS/400 user interface, see the article *An Integrated User Interface.)*

The expanded library search list facilities of OS/400 allow multiple national languages to be supported at the same time. The System/36 environment uses this support for displaying such items as messages, menus, and display formats. (For additional information, see the article *Software Design to Support National Languages*.)

Operator Control Commands

System/36 operator control commands, such as Status Print and Change Print, are used by programmers, operators, and general users to display and control jobs, spooled print files, and the like. In addition to providing a familiar command interface for System/36 users, the System/36 environment support allows menus containing operator control commands to be migrated to the AS/400 system. System/36 commands are supported by one of these techniques:

- Some commands are supported by specialized System/36 environment programs that provide similar functions to the System/36 commands. For example, the Status Session command calls a System/36 environment program that displays information, similar to that on System/36, about the current job.

- Some commands are supported by directly mapping the System/36 command to the corresponding AS/400 command. In many cases, the System/36 environment support was incorporated into the AS/400 support. For example, the Status Print command is mapped to the AS/400 Work with Spooled Files command. The information displayed is very similar to that on the System/36.

- Some commands are not supported directly. Users must provide information different than that on the System/36. In these cases, a message is issued to the user, instructing the user on how the task is performed on the AS/400 system. For example, the Change Print command results in a message describing how to cause one print file entry to print after another print file entry. After the user responds to the message, the AS/400 Work with Spooled Files display is presented and the user can change the spool file entries so the spool files print in the desired order.

  This method of giving instructions to accomplish the function with AS/400 commands is an easy way for users to learn the AS/400 commands. Once the user learns the names of these commands, the users can use the AS/400 command names directly, rather than the System/36 environment commands.

Message Support

The four types of System/36 messages (user, program, console, and subconsole) are available on the AS/400 system.

- User Message: A user message is sent by a user or procedure to either a work station or user by the MSG command. An example of this is "Peter, can you attend a meeting at 2:00 today?"

  The underlying message support of the AS/400 system is functionally very rich and flexible. OS/400 automatically creates a message queue for each user as he is enrolled, and for each work station or printer device as it is created. The user can modify the characteristics of these message queues. For example, a message queue on the AS/400 system can operate like a System/36, where it immediately notifies the user when a new message is received. The system also allows the user to hold messages

and not notify the operator, or to interrupt immediately what the user is doing and display the message.

- Program Message: A program message is sent by a program to a work station when an error is detected. The operator has a choice of ignoring the error, retrying, or canceling the function. An example of a program message is "Library LEDGER already exists."

  The System/36 environment program message support is designed to present messages like a System/36. Program messages sent by System/36 environment functions and applications are presented in the same format as they were on the System/36, with the same options, and the ability to return to the procedure prompter ($HELP function of System/36). System/36 automatic-response values for application messages will continue to function, and the user can take advantage of other automatic-response capabilities provided by the full AS/400 message support. The System/36 message manual is now online, and can be viewed without leaving the work station.

- Console Message: A console message is sent to the system console operator to manage system resources or batch jobs. An example of a console message is "Please insert diskette ABC in the diskette drive."

  The AS/400 system has a single system operator message facility designed to handle all environments. The AS/400 system operator message queue (QSYSOPR) is used like the system console message queue on the System/36. The System/36 commands and OCL statements for sending a message will send messages to QSYSOPR when the console is specified. The system operator message queue can be viewed by anyone from any work station. It is not restricted to a specific device as on the System/36. When a program message is sent to the system operator message queue, an informational message is also sent to those using that program, to inform them that the job is waiting for operator action.

- Subconsole Message: A subconsole message is sent to a subconsole operator who is managing a set of printers that are near the work area. An example of a subconsole message is "Please mount forms CHECKS into printer P2."

  As stated earlier, each printer has a message queue. The operator responsible for managing a printer or set of printers can use the Work with Writers command (similar to the System/36 Status Writer command) to view all of the active printers. If the status of the printer indicates it is waiting because of a message, the operator can then display the messages for that device and correct the situation.

Menu Support

System/36 environment menu support is an extension of System/36 menu support and is integrated into OS/400. User menus for both the System/36 environment and the AS/400 system consist of a display format and a message file (message member on System/36). The System/36 interface for creating menus is supported (FORMAT, CREATE, BLDMENU, and SDA procedures), in addition to the AS/400 methods for creating menus (Create Display File, Add Message Description, and Start Screen Design Aid commands). The System/36 interfaces to display a menu (Menu operator control command and MENU OCL statement) are supported by the System/36 environment, similar to the support offered on System/36. In addition to displaying user menus, the System/36 environment has been enhanced to display AS/400 system menus. The AS/400 system menus guide the user in performing system tasks, in the same way as the help menus provided on System/36.

The operational characteristics of menus have been changed from System/36. For example, on System/36, the Dup key was used to retrieve the last function only, while the AS/400 system can retrieve any previous function. The user can display and select from all of the functions that were entered on a menu during the current session.

System Request/Attention Key

The system request support on the AS/400 system is combined with the inquiry (attention) support from System/36. The support is a common interface that is tailored to the environment the user is currently operating in. This merging of the functions was accomplished with these changes:

- The System Request key shows the System Request menu from any signed-on display station.

- The System/36 system request function to display the messages sent to the system operator is an option on the System Request menu.

- The options are tailored to the current operating environment. For example, if a System/36 environment job is running, the Display current job option shows the status of the job using the System/36 environment Status Session command. If an AS/400 job is running, the Display current job option shows the status of the job using the AS/400 Display Job command.

Unlike the System/36, the Attention key is not reserved for use by the system. The System/36 environment user can use the support available to

all AS/400 users to define a program to be run when the Attention key is pressed. This attention key program can be defined in the user's profile or by issuing a command (Set Attention Program).

Application Interface
A major design consideration was to ensure that the primary application interfaces on System/36 (RPG II, COBOL, OCL, utilities, and so forth) are supported by the System/36 environment. The System/36 environment was designed so information produced by these applications, and the interfaces seen by the users of these applications, are equivalent to that of the System/36.

Languages
The language heritage of the System/36 began on the System/3 and has evolved and grown to meet the ever-expanding interactive processing, work station, and communications requirements of the data processing community.

For example, on the System/3, communications was principally batch-oriented and was accessed using the telecommunications specification. The System/32 supported a small built-in display that was six-lines long and 40-characters wide. RPG II Keyboard, Console, and CRT file specifications were added to accommodate accessing those devices. System/34 added the concepts of MRT programs, NEPs, no requestor-terminal programs, read under format, and SSP-ICF operations. System/36 in turn added work station file specifications to allow for a data dictionary specification for externally defined SSP-ICF formats.

To protect application investments and to provide an easy migration path for System/3, System/32, System/34, and System/36 customers, the System/36 environment RPG II and COBOL compilers have maintained all of these language extensions. System/34 and System/36 are not strongly data typed. Users can leave blanks in a numeric field and the System/36 would treat that as zero and allow arithmetic operations on the field. The base instruction set of the AS/400 system supports the stronger data typing of RPG III, COBOL'85, PL/I, and BASIC and will detect a decimal data error if the user attempts an arithmetic operation on a field containing non-numeric data. Extensions to the base instruction set allow it to operate similar to the System/36 if a decimal data error is encountered when performing zoned arithmetic. The System/36 environment functions also provide additional support for System/32, System/34, and System/36 language extensions, so RPG II and System/36 environment COBOL programs must be processed within the System/36 environment.

AS/400 programs (RPG III, COBOL'85, CL) can also run in the System/36 environment. Because the System/36 environment is an integrated part of OS/400 and has all of the facilities of the operating system available to it, an RPG III program runs in the System/36 environment without using the System/36 environment-sensitive functions.

In addition to the functions currently available to the System/36 application developer, the System/36 environment compilers provide expanded capabilities. These items, for example, are available to RPG II programmers:

- Greater than 64K program size
- Maximum number of arrays is increased from 75 to 200
- Ability to call any other program on the system
- Maximum number of files used by a program increased from 20 to 50

To enhance programmer productivity, the System/36 environment supports the full AS/400 debugging facilities for RPG II and COBOL. (For more information on the System/36 environment compilers, see the *Application Development Support* article.)

Operation Control Language
A key part of any System/36 application is the procedures the programmer has developed to control the flow of programs within the application.The AS/400 system supports both System/36 OCL within the System/36 environment, and compiled AS/400 CL, which is syntactically quite different from the System/36.

A programmer can include CL programs and AS/400 commands in a System/36 procedure. This allows a programmer to access new functions or facilities provided by OS/400, without having to rewrite the System/36 procedures as CL programs.

The System/36 environment OCL reader and interpreter supports:

- The individual OCL statements themselves.
- Procedure control expressions that allow the user to build conditional logic into the procedure. The types of functions available include testing for a file's existence, performing simple mathematical functions to control iterative operations, or checking the volume ID on a diskette.
- Substitution expressions that allow the user to extract data from the system and incorporate it into an OCL statement. For example, substitution expressions are available to request the user ID of the person initiating the procedure, the current system time or date, and the value of any parameter passed to this procedure.

To facilitate intermixing OCL statements and CL commands in a procedure, a few new OCL substitution expressions have been added to the System/36 environment. The new substitution expressions are provided to assist the programmer who wishes to add CL to a procedure and needs to provide the library name where a System/36 file is located or to be able to detect messages returned by a CL command.

Utility Support

On System/36, utilities perform basic tasks for users. The System/36 environment has a set of utilities very similar to the System/36 utilities. System/36 environment utilities have the same appearance and issue the same messages as the System/36 utilities.

Several techniques support System/36 environment utilities:

- Some utilities are supported using AS/400 commands to perform function similar to System/36. For example, the $COPY function to display selected records from a diskette file uses three AS/400 commands to perform the single function requested by the user. The Restore Object command restores the file from diskette, the Copy File command selects the records, and the Display Physical File Member command displays the records.

- Some utilities are supported by System/36 environment programs. For example, $SETCF, which sets default values for a display station, calls a program that sets corresponding values in System/36 environment control blocks.

- Some utilities are supported by displaying an AS/400 menu. The user can select options from the menu to perform the desired function. For example, $CNFIG, which defines the devices attached to System/36, displays the AS/400 Configuration menu.

- Some utilities are supported by AS/400 command prompts. For example, $PRUED, which updates a user profile, provides prompts that request the parameters for the AS/400 Work with User Profiles command.

- Some utilities are not required on the AS/400 system because the architecture of the AS/400 system is different from System/36. The utility control statements for these utilities are checked for syntax and then ignored. For example, because the AS/400 system does not require that a file occupy contiguous disk space, $FREE, which groups unused disk spaces together, is checked only for syntax. This syntax checking benefits users developing applications to run on System/36.

MRT Support

The System/36 environment supports MRT applications, which have been written to manage and process requests from multiple devices. MRT requesters can be either interactive jobs or communications jobs. An MRT is initiated when the first requester calls it, and the MRT remains active as long as at least one requester is using it. An MRT may also be an NEP. An MRT that is not an NEP ends when it has no requesters using it. An NEP MRT with no requesters remains active, waiting for the next requester. MRTs are used extensively in System/36 applications for both storage and performance reasons.

MRT support is critical for source-level compatibility of System/36 applications, and this support is provided by the System/36 environment. The System/36 environment uses AS/400 work management support to initiate the MRTs, AS/400 data management support to pass the requester devices into and out of the MRTs, and relies heavily on the event support of the AS/400 system for interprocess communication.

The MRT is implemented as an AS/400 user job with the same structure, attributes, and capabilities of other user jobs, but with a special attribute that identifies it as an MRT. This allows the MRT to be controlled through the same job control interfaces as other AS/400 jobs, in addition to being controlled through the System/36 job control interfaces. Various AS/400 job control displays identify an MRT as having a job type of MRT; indicate if an MRT is also an NEP; display the maximum number of users allowed in an MRT (MRTMAX); display the number of users currently in an MRT; and indicate if a System/36 environment job is currently routed to an MRT and, if so, indicate the name of the MRT.

The System/36 environment default value for MRT security is the same as it is on the System/36. In addition, a customer has the option of choosing other levels of MRT security to reduce the number of authorizations needed when enrolling new users on the system, and to improve the performance of routing users to active NEP MRTs.

Data Base Support

The AS/400 data base contains support for some System/36 concepts. For example, the System/36 file attribute that prevents the records in a file from being deleted was generalized by AS/400 data base support into attributes preventing input, output, update, or deletion of records in a file. (For more detailed information on the advantages of the AS/400 data base support and how it supports files migrated from System/36, refer to the article *An Integrated Data Base*.)

The System/36 environment provides the additional support necessary to allow most System/36 applications to function like they did on System/36, while using the AS/400 data base

support. The System/36 environment supports the OCL FILE statement (which is required for every disk file used in a System/36 application) by mapping the parameters to the equivalent AS/400 data base functions, and allocating any files indicated by the FILE statement.

The System/36 environment supports improved performance when an application uses the same data base file in consecutive programs called from the same procedure. This is a fairly typical scenario because the 64K program size limit on a System/36 often results in splitting an application into many small programs, each opening the same file. The System/36 environment keeps data base files open after a program has completed. If the next program uses the same file, the System/36 environment connects that program to the open file. If the next program does not use the same file, it is closed.

The System/36 environment data base support takes advantage of the additional function available on the AS/400 system. For example, the limit on the number of open files has been increased significantly, thus allowing a single program to perform file updates that would have required multiple programs on a System/36. Another advantage is the AS/400 disk management capabilities. A file does not have to be located in contiguous storage locations and space is not reserved until it is needed. Users also have access to the data integrity and recovery functions of the AS/400 data base.

Display and Communications Support

The System/36 environment provides support for System/36 applications that interact with any combination of display devices and communications devices. Functions necessary for System/36 compatibility that are not part of AS/400 data management are incorporated as extensions of AS/400 data management and are only used when an AS/400 application is running.

The basic support and structure of AS/400 data management for display devices was adopted from System/38. System/36 display formats are migrated to AS/400 display device files. In addition, OS/400 Intersystem Communications Function (ICF) was incorporated into the AS/400 data management structure. The concept of ICF, a generalized high-level interface for communications applications, was adopted from System/36. Support was incorporated into AS/400 data management for ICF files, which are used for I/O to all types of communications devices supported by the AS/400 system. System/36 communications formats are migrated to ICF device files. (For more information on AS/400 data management support, see the article *A Structured Approach to Data Management*.)

Major capabilities were built into the System/36 environment to make migration transparent to most System/36 applications. The structure (work areas) of System/36 data management is organized to support programs that issue I/O operations directly to devices. The structure of AS/400 data management is organized to support programs that issue I/O operations through a device file to a device. On the AS/400 system, multiple device files can be open at the same time, and work areas representing a program's use of a device through a particular file are needed on all I/O operations. Information stored in the work areas on an output operation is required to successfully process the next input operation. No comparable requirements exist on System/36, as all information required to complete an I/O operation is associated with the program and the device, and not with the use of the device through a particular file. The System/36 environment masks these differences to provide support for System/36 applications, including the support for read under format.

A System/36 program can do I/O through both display formats and communications formats and can simultaneously wait for an I/O response from multiple display devices and multiple communications sessions. The AS/400 system does I/O through formats contained in a device file. The System/36 environment allows an application to issue a single input operation (Accept Input) to a display file (with one or more display devices attached) and an ICF file (with one or more communications sessions attached). The operation is satisfied by the first I/O response to complete.

Finally, the System/36 environment allows System/36 applications to use System/36 two-character device names instead of AS/400 10-character device names. The System/36 environment device name mapping takes into consideration the system-level device name information, as well as application-level device name mapping provided through OCL statements (WRKSTN and SESSION). The System/36 environment determines the appropriate device name mapping and passes it to AS/400 data management using generalized device-name mapping interfaces before calling a System/36 application program. Therefore, AS/400 data management is not aware that it is working with System/36 device names.

Read Under Format

To provide compatibility for System/36 applications, the System/36 environment supports read under format, which allows a System/36 program or procedure to read a format that was displayed by a previous program. The program can read through a different format and a different file than that used on the output operation. (The application program doing the read must know how to process the data it is receiving.) Read under format allows the user to enter data on the display while the second program is initiating, thus improving overall response time.

Read under format applies to all requester devices. A requester display device is the device through which an interactive user signs on. A requester communications device is a communications session through which a communications job was initiated by a procedure start request coming across a communications line.

The System/36 environment supports read under format by keeping display files and ICF files open after a program has attempted to close them. When a program opens the same file that was already opened by a previous program, the System/36 environment connects that program with the file kept open from the previous program. The second program then has the capability to read a format that was displayed by the previous program. When a program opens a different display file, the System/36 environment determines if read under format is in progress and, if so: intercepts the first input operation from the current program; issues the input operation through the old file; intercepts the completion of that input operation; moves the data to the input buffer of the new file; sets appropriate return codes; and sends the response to the application just as if the I/O had completed through the new file. When a program opens a different ICF file, the System/36 environment passes the communications session to the new file without disturbing the conversation in progress, using specialized ICF functions provided specifically for this purpose. This same level of support is provided for both synchronous and asynchronous input operations issued by System/36 applications.

Some additional complexities are introduced when supporting read under format between MRTs and single requester terminal (SRT) programs. The System/36 environment uses specialized functions provided by AS/400 data management to support the completion of an I/O operation in a different job from which the operation was started.

In addition to supporting read under format, keeping display files and ICF files open across programs provides significant performance advantages.

The System/36 to AS/400 Migration Aid

To assist the user in migrating from a System/36 to the AS/400 system, a menu-driven Migration Aid leads the user through the migration process, including the migrating steps that occur on the System/36 and those that occur on the AS/400 system.

The System/36 part of the Migration Aid summarizes what is on the system; identifies functions that must be changed to run on the AS/400 system; selects items for migration (such as libraries or office user profiles); moves the selected items to tape or diskette; and generates reports (items selected, saved, failed).

The AS/400 part of the Migration Aid restores all migrated items; enrolls migrated users; compiles RPG and COBOL programs from the source code; creates message files, display files, and menus from saved source; converts saved device configurations, data dictionaries, documents, and office objects (calendars, mail logs); and generates reports (objects migrated successfully and unsuccessfully). See Figure 1 for a list of the items migrated, and the AS/400 object to which they are converted.

Coexistence Support

Coexistence refers to the fact that for many customers, their AS/400 system will coexist with other systems. Many users will use the System/36 environment as the central site for developing and testing applications that will run on one or more System/36s. This allows them to take advantage of the AS/400 programming productivity features, such as interactive debugging.

Applications that work in the System/36 environment also work on a System/36. In cases where the System/36 environment provides more support than the System/36 (for example, the number of files that may be opened in a single program), warning messages are issued as the application is compiled. In this way, users who do not plan to move applications back to a System/36 are allowed to expand their applications to take advantage of the AS/400 support, and users who plan to move applications back to a System/36 are warned whenever they use a function that will not run on a System/36.

Applications and data can easily be moved through a network of System/36 and AS/400 systems using either communications facilities or save-and-restore support. In addition, applications and data can be moved to the AS/400 system from the System/34 and System/32.

AS/400, System/36, System/34, and System/32 users can use the standard save-and-restore functions on their respective systems when exchanging applications and data. In the past, migrating to a different architecture required that the data be converted before migration, usually into a data interchange format. This is no longer necessary because the AS/400 system recognizes and generates the internal formats of applications and data used by these other systems. The three advantages to this approach are: the users of the respective systems do not need to learn new save-and-restore interfaces; saving a large file to tape using standard System/36 save procedures is approximately six times faster than saving the same file in data interchange mode; and archived media (tapes and diskettes) can be directly migrated to the AS/400 system without first having to restore them to the system from which they were saved.

Conclusions

The System/36 environment provides AS/400 support for System/36 applications and users. The System/36 environment provides a high degree of source-level compatibility for System/36 applications. This includes support for APIs such as RPG II, COBOL, procedures, OCL, utilities, menus, commands, messages, display formats, and communications formats. End-user interfaces for accessing System/36 functions are supported and mapped to corresponding AS/400 functions.

The System/36 environment consists of operating system extensions that were designed to provide System/36-equivalent function, using the underlying support of the AS/400 system wherever possible. This approach results in performance for System/36 applications that is equivalent to that available for AS/400 applications, and enables the System/36 environment applications and users to have access to the new functions of the AS/400 system.

Users may easily migrate most applications and data from a System/36 to the System/36 environment, as well as from the System/36 environment back to a System/36. This allows the System/36 environment to serve as a growth path for existing System/36 users, as well as for developing central-site applications that will run on a System/36. In addition, users may choose to gradually rewrite their System/36 applications to take advantage of new AS/400 functions. Users content with the function provided by System/36 applications can continue to run in the System/36 environment, while obtaining the performance benefits of the AS/400 system.

Acknowledgments

We would like to acknowledge the contribution of Guy W. Vig and Michael P. Anderson in supplying information relative to the utility support, System/36 to AS/400 Migration Aid, and coexistence support sections of this article.

The Communications and Networking Structure

Describes the data communications hardware and software structure in the AS/400 system and discusses how it supports today's function while laying the foundation to meet future requirements.

James O. Walts and Paul R. Mattson

Introduction

The interest in and use of data communications and networking facilities has grown dramatically in recent years. Part of this growth has been driven by market demand, while part has been driven by technology.

The information managers in business and industry recognize that their information is a valuable corporate resource. What information will be collected and how it will be managed and used has become very important. Getting accurate information to the correct places in a timely fashion for decision makers to take action has become an integral part of businesses' challenge to remain competitive.

Data communications plays a significant role in meeting these challenges. Business has become more and more dependent on its data communications facilities as these challenges are met. In many cases, even the very way business is carried on has changed due to emerging data communications technologies. As a result, a growing demand exists for functionally rich, reliable, and manageable data communications functions, products, and facilities.

In a complementary way, technology has contributed to the growth in data communications products and services. Increasing processing power and storage capabilities at lower and lower costs have allowed new applications that were once prohibitively expensive. Existing function is enjoying new levels of performance for the price.

The AS/400™ system features many of the capabilities that have been driven by the data communications market demands, including the AS/400 implementation for various communications protocols. Some of these features are common application facilities, Systems Network Architecture (SNA), management services, and separate input/output (I/O) processors. The AS/400 system delivers these functions today by integrating advanced hardware and software technologies into an overall structure designed for functional expansion tomorrow.

Design Objectives

The AS/400 data communications structure was designed with a number of goals in mind. First, the structure had to provide comprehensive functional capability at a competitive price-for-performance level. In addition, the data communications structure had to support multiple architectures in a flexible and extendible fashion, by supporting multiple, concurrent data communications architecture implementations and the sharing of physical resources where meaningful. It had to have an extendible common framework, within which various communications protocols could be implemented. The various communications protocols had to be presented to the application in a consistent high-level fashion, thus shielding the application writer from much of the protocol detail. And, the structure must maximize the ability of the AS/400 system to communicate and operate with other IBM and non-IBM products today and in the future, including the rich complement of SNA capability, asynchronous communications support, binary synchronous communications support, as well as affinity with the emerging open systems interconnection (OSI) architectures. And finally, the structure had to allow the system and data communications operator to configure networks easily, check their status, and monitor their behavior. The data communications operator must be able to get maximum utility from the network with minimal management effort. The structure of AS/400 data communications was designed to meet these objectives. An AS/400 sample network is shown in Figure 1.

Data Communications Structural Overview

The AS/400 data communications structure can be viewed as a two-dimensional matrix. Each cell within the matrix provides a particular communications function. (Figure 2 shows this communications matrix.)

The vertical dimension of the matrix shows the distribution of **architectural layer functions** (such as application, presentation, and session functions). AS/400 function has been distributed across the System Processor, the I/O processors, and physical hardware attachments. Function is distributed throughout the system, depending on such parameters as sharing architectural layers, sharing physical hardware, and performance.

The horizontal dimension shows various **communications protocol implementations** (such as SNA, asynchronous, and binary synchronous protocols). This horizontal dimension depicts the integration of dissimilar architectural

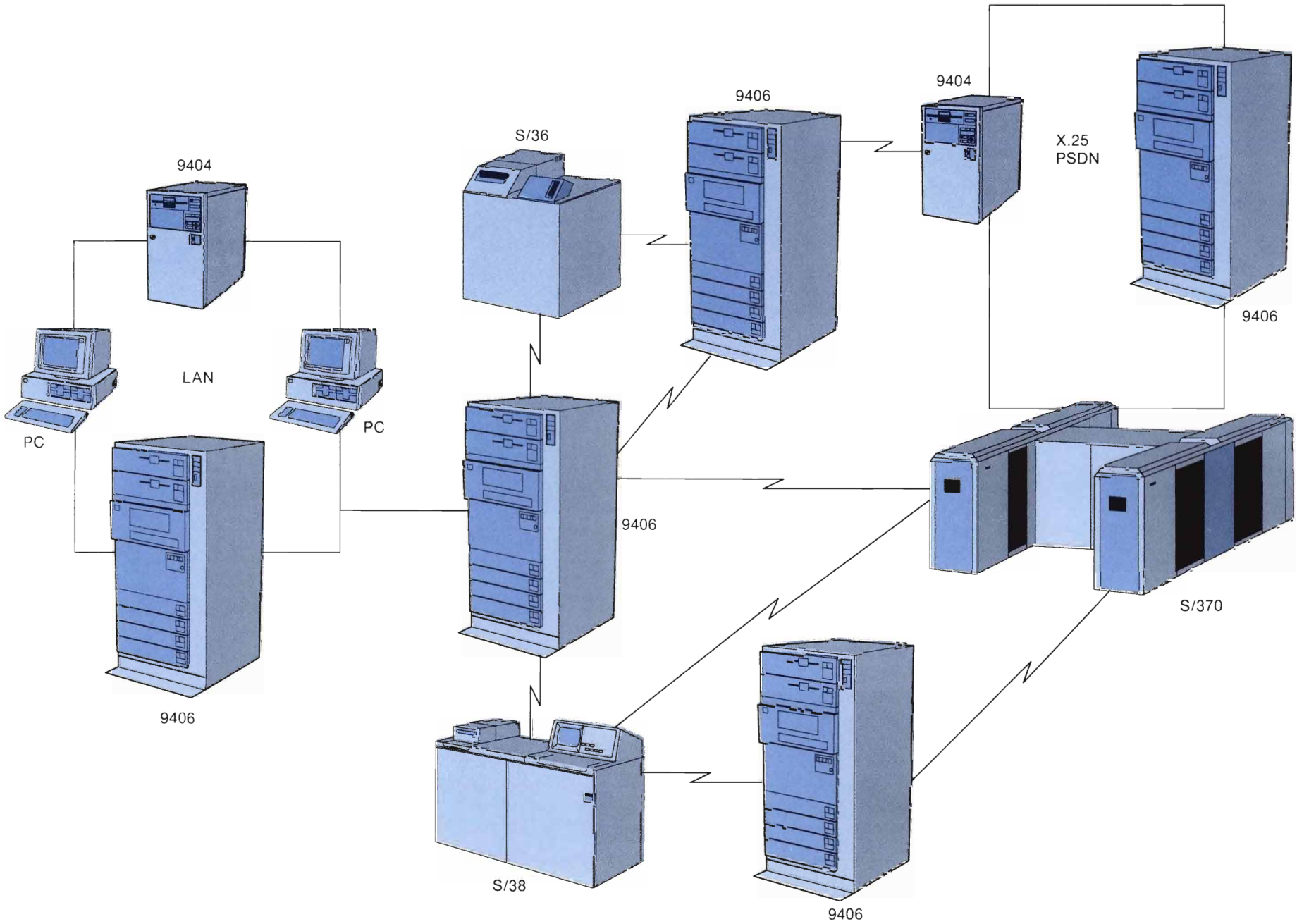


Figure 1 AS/400-System/370 Sample Network

implementations into the same structural layers of the system. It also shows sharing common system functions and packaging at several of the vertical layers (for example, a common communications I/O processor is available for all protocols).

This matrix structure is presented to the user through management services and common application facilities. Common application facilities provide the user with consistent access to communications functions. These facilities are common to all protocols, thus providing a uniform interface. Management services permeate all structural layers of the system. In this way they can control and monitor all communications functions. The hardware and software, which is self-defining and self-diagnosing, aids the operator in network configuration, interrogation, and monitoring.

Data Communications Relationship to SNA and OSI Models

Figure 3 provides a composite view of the components of the initial AS/400 data communications offering. The overall AS/400 implementation is shown with a comparison of SNA and OSI implementations. The figure shows the details of how the functional layers of SNA have been implemented in the AS/400 system and how the physical, data link, and network layers of OSI have been embodied in the AS/400 system. As an example, IBM Token-Ring Network and X.25 protocols serve as alternative data link controls for the SNA Path Control function. Also, independent protocol implementations can share a physical resource. As an example, the AS/400 system can communicate with an ASCII host system and an SNA host system concurrently over the same X.25 physical port. The ability to share physical resources is important to satisfy the second AS/400 design objective of supporting multiple architectures in a flexible manner.

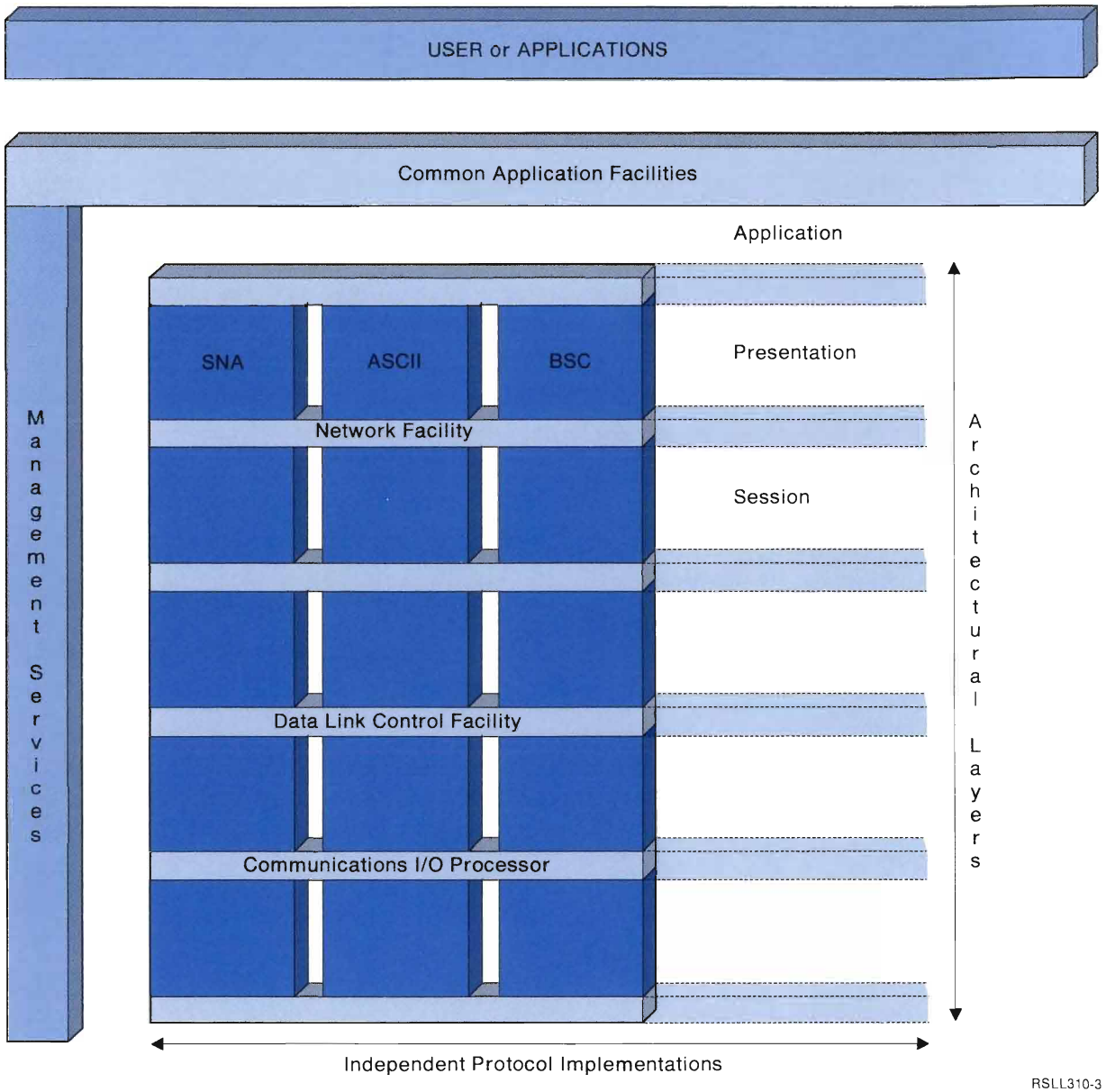


Figure 2 Two-Dimensional Matrix of the Data Communications Structure

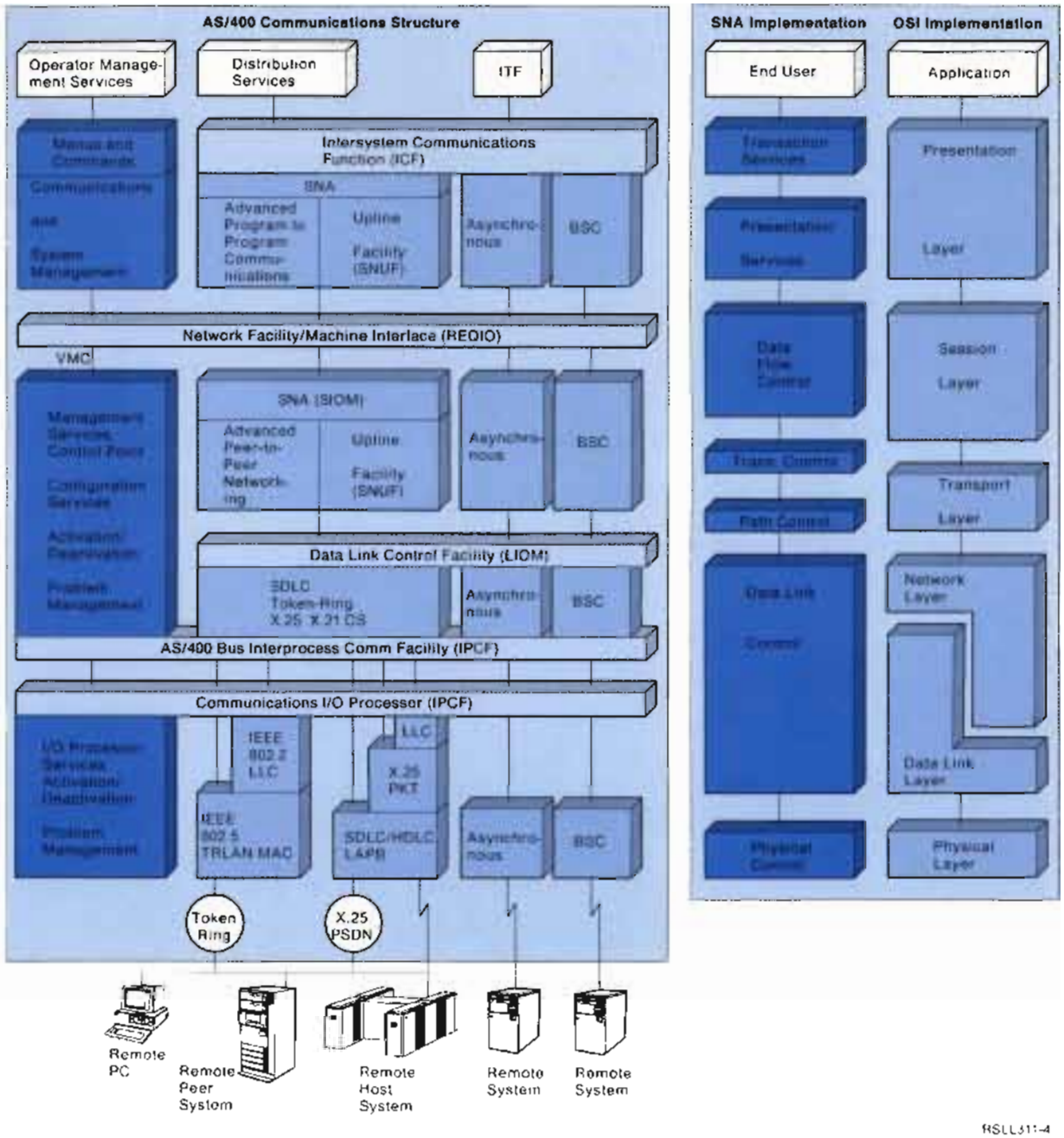


Figure 3 Components of AS/400 Data Communications as Related to SNA and OSI Implementations

The vertical dimension of the diagram shows the functional distribution across the architectural layers from the operator or application through the machine interface to the I/O processor, and then through the physical network interface. The diagram shows the AS/400 structure that reduces the complex data communications environment into smaller isolated functions. The intersystem communications function, network facility, data link control facility, and interprocess communications facility are well-defined internal interfaces that enable sharing of the various architectural layer implementations. Additionally, these well-defined interfaces allow for movement of functions within the system, allowing the AS/400 system to take advantage of technology as it becomes available. That is, a function can be distributed to a different layer in the system in a manner transparent to the user. This means that a series of compatible systems can be built with varying functional distributions without changing the user interfaces (this is not specific to communications, but is a general AS/400 structural feature).

Management Services

As shown in Figure 2, management services span all functions in the AS/400 system, from the system operation and the physical hardware connections to the communications network and the local devices. It provides the facilities used to manage the system complex and the network within which it exists. These **management services** are known as Communications and Systems Management (C & SM), the parts of which are shown in Figure 3. C & SM fully integrates *both* the communications and local systems management into Operating System/400™ (OS/400™) through operator menus, commands, and the common applications interface.

Operator Menus and Commands. The AS/400 system presents to the system operator an extensive set of menus and commands to use C & SM. The operator can manage the system

resources through menus that provide the ability to gather problem data (called alerts) into a data base, monitor the network, configure the network, and perform problem determination from a central point. The alerts are automatically sent to the central point from anywhere in the network. The central point can be an AS/400 system and a System/370 using the NetView™ Distribution Manager. (For more details, see the article *Electronic Customer Support.)*

Management Services Control Point. The management services control point consists of configuration services, activation and deactivation services, and problem management. The management services control point manages mapping system communications objects to physical resources. It is the cornerstone of communications resource management and coordinates creating all System Processor and I/O processor tasks that provide the communications support from the user application to the physical port. The support selected is a function of the configured communications objects. It includes activating the appropriate I/O processor-based tasks for communications objects being varied on and creating appropriate data link control facility and network facility tasks based on the line and controller descriptions, respectively. It maps application requests for sessions to the network facility that supports those sessions.

The management services control point is also responsible for alerting the AS/400 system of changes in physical resource status. This occurs during the coordinating process required to activate a communications resource, as well as during error conditions and user-initiated deactivation procedures. In addition, the management services control point orchestrates communications second-level recovery. That is, after a device, controller, or line is declared inoperative, the management services control point coordinates the applications, the operating system, and the system operator to re-establish or end the communications path in the system. Management services also coordinate the control point functions of advanced peer-to-peer networking (APPN). (For more information on APPN, see the article *Advanced Peer-to-Peer Networking*.)

Configuration Services: Configuration services provide a set of facilities for controlling and maintaining configuration information for the AS/400 system. The AS/400 configuration is integrated into the system and is object-oriented. That is, composite data structures are defined at the machine interface on which only well-defined functions may operate. Several configuration objects are defined. These include the line description, the controller description, the device description, the mode object, and the class-of-service object. These objects may be created, examined, varied on and off, changed, or deleted while the system is fully operational.

The line description is a definition of the way in which a communications port is to be used. It is through the line description that one selects the particular data link protocol to be used and the physical and logical parameters that condition that protocol's behavior as it communicates with a remote system. The line description specifies the resource name that corresponds to a particular communications port on the physical communications adapter. The actual communications port is bound to that software object at the time the line is varied on. This late binding contributes to a number of features. The adapter may be moved to different hardware slots without software re-configuration. Because the validity of resource names is not checked until vary-on time, central-site development and testing of configuration programs is made easier. Several different objects may call for use of the same port. This mutual exclusion is managed by the system at vary-on time. The controller description defines the characteristics of the remote system and the session-level LU-type protocols. The device description defines the physical or logical device to which sessions are to be established.

Activation and Deactivation Services: Activation and deactivation services provide I/O processor management services to the system. The management services control point calls on the activation and deactivation services to activate appropriate I/O processor microcode tasks based on the line description; this includes loading the multitasking I/O processor with the selected protocol's re-entrant microcode.Therefore, only the first line on an I/O processor of a particular data link control type requires a microcode download. Activation and deactivation services customize the activated microcode task by passing configuration and recovery information to it. Activation and deactivation services are also involved in the deactivation process that occurs when a line description is varied off.

Activation and deactivation services provide a focal point for I/O processor-detected errors. I/O processor hardware and software events are passed, in well-defined formats, to activation and deactivation services where they are logged. The system command, Work with Error Log, and problem analysis functions are interfaces to this log.

Activation and deactivation services use the threshold concept to alert the operator of behavior outside the acceptable operating bounds as it occurs. This feature helps the operator identify problems negatively affecting network performance.

Activation and deactivation services provide the interface to the I/O processor for debugging. The debugging functions include setting breakpoints, dumping I/O processor storage, and tracing the paths taken by I/O processor microcode. These tools are for use by service representatives.

Problem Management: Problem management assists C & SM in problem determination, problem diagnosis, and configuration monitoring. The problem management interfaces work with the I/O processor services to perform activation and deactivation of network resources, to test network resources, and to collect statistical information on the network resources. Problem management then reports the resulting system reference codes (SRCs) for each of these operations to the C & SM facilities. These SRCs are processed by C & SM. If the situation cannot be handled locally, it is reported to the focal-point problem management system through the use of C & SM generic alerts facilities. These functions provide the ability to manage the system from a central site.

Intersystem Communications Function
Intersystem Communications Function (ICF) provides the common application facilities shown in Figure 2. In Figure 3, the various protocol implementations (SNA, asynchronous, and binary synchronous) appear below ICF. ICF presents a common application interface for easily accessing these communications implementations. The communications protocols are implemented in the vertical dimension shown in Figure 2. Therefore, ICF is a consistent interface across all protocol implementations. This common application interface shields the end-user application from the detail of each individual protocol. It allows the application programmer to define the application data externally to the program and independently of the protocol type. The specific protocol is selected according to the configuration.

The AS/400 system has several of its own system applications that are written to the ICF interface. One such application, SNA distribution services (SNADS), provides a set of asynchronous services consisting of queueing, safe storage, and scheduling services, which support distribution of a variety of data objects. Examples of other IBM applications are distributed services node executive (DSNX) and interactive terminal facility (ITF).

Networking Facilities/Machine Interface
The machine interface contains a set of instructions for accessing all network facilities. These instructions provide the means for configuration, activation and deactivation, I/O, and problem handling. The instructions operate together to provide a group of control point services that set up an optimal route for delivering application data with integrity and security, for supporting network management and for providing network recovery. These functions are provided by the station I/O manager (SIOM) in conjunction with the management services control point. The request I/O (REQIO) machine instruction is the main instruction for performing data transmission and reception. This instruction is processed by the station I/O manager shown in Figure 3. The station I/O manager provides the capability to share, on a session basis, the resources of a remote system. It multiplexes the data for a set of sessions to a data link control facility. It also provides session-level error detection and recovery on behalf of the application. The station I/O manager is established by the management services control point during controller description vary-on processing. The station I/O manager task, based on the controller description and device description, provides a particular LU-type protocol service for an application.

Data Link Control Facilities
Data link control facilities provide SNA and non-SNA (refer to Figure 3) applications with a common interface to the components that deliver the data to the adjacent system in the network. They are designed to transparently multiplex several different station I/O managers to a single physical port that supports logical adjacent links, for example, X.25 and IBM Token-Ring Network. As an example, the X.25 data link control can concurrently multiplex the following dissimilar communications environments to the same physical X.25 port: the AS/400 system as a secondary SNA station role when communicating with the host System/370; the AS/400 system as a primary SNA station role when communicating with a remote personal computer; and the AS/400 system as an asynchronous pad station when communicating with an asynchronous host system. This high level of concurrency of communications environments provides maximum use of the hardware.

The line I/O manager (LIOM) is shown in Figure 3 under the data link control facility box. The line I/O manager provides a transparent interface to the station I/O managers independent of the underlying data link protocol and network being used. It is this transparency that allows the addition of new data link controls into the structural matrix without affecting those types of station I/O managers that currently exist. In a complementary way, new session and transport layer implementations can be introduced to share the existing physical network support. The line I/O manager is put in place by the management services control point during line description vary-on processing. It manages the physical link-level activation for the management services control point, multiplexes a set of station I/O managers onto a single physical port, and participates in the second-level line recovery as directed by the management services control point.

I/O Processor Facilities
The AS/400 I/O processor is a general purpose processor that is attached to the System Processor through the system I/O bus. (For more information, see the article *The Multiple-Function Input/Output Processor*.) Its purpose is to off-load support of I/O interfaces and their associated protocols from the System Processor.

The I/O processor provides a multitasking operating system and management functions that allow it to support a number of communications ports concurrently. For each port, a set of protocol support tasks are put in place at the time the line is varied on. This facilitates the efficient use of I/O processor storage and processor resource. A number of data link and physical controller tasks are available in the AS/400 system (IEEE 802.2, X.25 packet-switching digital network, synchronous data link control, asynchronous, and binary synchronous).

Each set of I/O processor protocol tasks has a corresponding line I/O manager task in the System Processor. The connection between the line I/O manager task and the I/O processor protocol task (IPCF) provides a full duplex and queued message-based service. This service masks most of the details of the bus structure and physical I/O processor card addressing from the line I/O manager. That is, it provides a location-independent service to the line I/O manager and, in doing so, also provides a transport mechanism independence. This allows for repackaging hardware and changing function distribution without upsetting the line I/O manager design or any of the I/O design above it.

Within the I/O processor, the functions provided are also distributed across a number of tasks. This distribution is based on the SNA and OSI implementations. At each layer within the I/O processor, connections are established to the System Processor, allowing the System Processor to take advantage of any of the exposed layers shown in Figure 3. This is key to sharing the same physical port while using different upper-layer protocol services.

The AS/400 I/O processors have provided a means to move compute-intensive operations (such as data link controls) out from the System Processor. This relieves the System Processor from those burdens, allowing for efficient I/O processor microcode implementations and greater overall system throughput.

Conclusions

The AS/400 communications structure provides a distinct environment for integrating dissimilar communications protocols into the operating system. This structure allows the designer to concentrate on the protocol function rather than how to accommodate the protocol in the system. The result is a well-integrated set of dissimilar communications protocols.

The AS/400 communications and networking structure supports all the functional capabilities of preceding products, as well as providing the structure on which to expand its initial offering. Through the common applications facilities, new protocols can be integrated and new hardware attachments added without disrupting an investment in communications applications software. In addition, a rich set of communications architectures is provided from which to build networks, while offering easily accessed complex environments. Management services provide the operator functions necessary to efficiently administer the communications facilities.

The AS/400 system provides an integrated communications hardware and software solution which is designed to grow with tomorrow's needs.

Advanced Peer-to-Peer Networking

Describes the implementation and advanced networking features that enhance system-to-system and program-to-program communications.

Raymond K. Harney and Christopher H. Jones

Introduction

The expanding AS/400™ telecommunications market requires networks built with low-cost systems that are able to grow and participate with existing IBM Systems Network Architecture (SNA) networks. In addition, allowing distribution of resources among different processors without requiring end users to be aware of the physical location of these resources is central to the usability of a distributed operating system. This transparency of network location and the physical medium used to gain access to these resources will be an integral part of corporate telecommunications strategies as the networking environment grows during business transitions of the 1990s and beyond.

In March of 1987, the Low-Entry Networking architecture was announced as the strategic networking element for common communications support in the Systems Application Architecture™ (SAA™) strategy. The System/36 and System/38 combine the verb set and application program interface that is advanced program-to-program communications/logical unit type 6.2 (APPC/LU type 6.2), with the Low-Entry Networking, or node type 2.1 transport layer functions, into product implementations called APPC. The AS/400 system has built upon this implementation of the Low-Entry Networking architecture with the development of advanced peer-to-peer networking (APPN) to meet growing distributed processing requirements. Advanced functions are offered, such as distributed directory searches, dynamic route selection, and intermediate session routing based on transmission priority. (For early networking requirements in an intermediate system environment, see Baratz et al [1].)

AS/400 APPN support allows applications written for the APPC/LU type 6.2 application program interface to communicate with remote partner applications without modification when multiple AS/400 systems are providing networking services. In addition to providing networking services for AS/400 users, other systems that implement only the base Low-Entry Networking architecture will also be able to use these services. (For a list of IBM systems that have implemented the Low-Entry Networking architecture, see Sundstrom et al [2].)

The Evolution of APPC and APPN

In 1983, IBM introduced an SNA peripheral node type, node type 2.1 (or as it is known today, Low-Entry Networking) that supports point-to-point

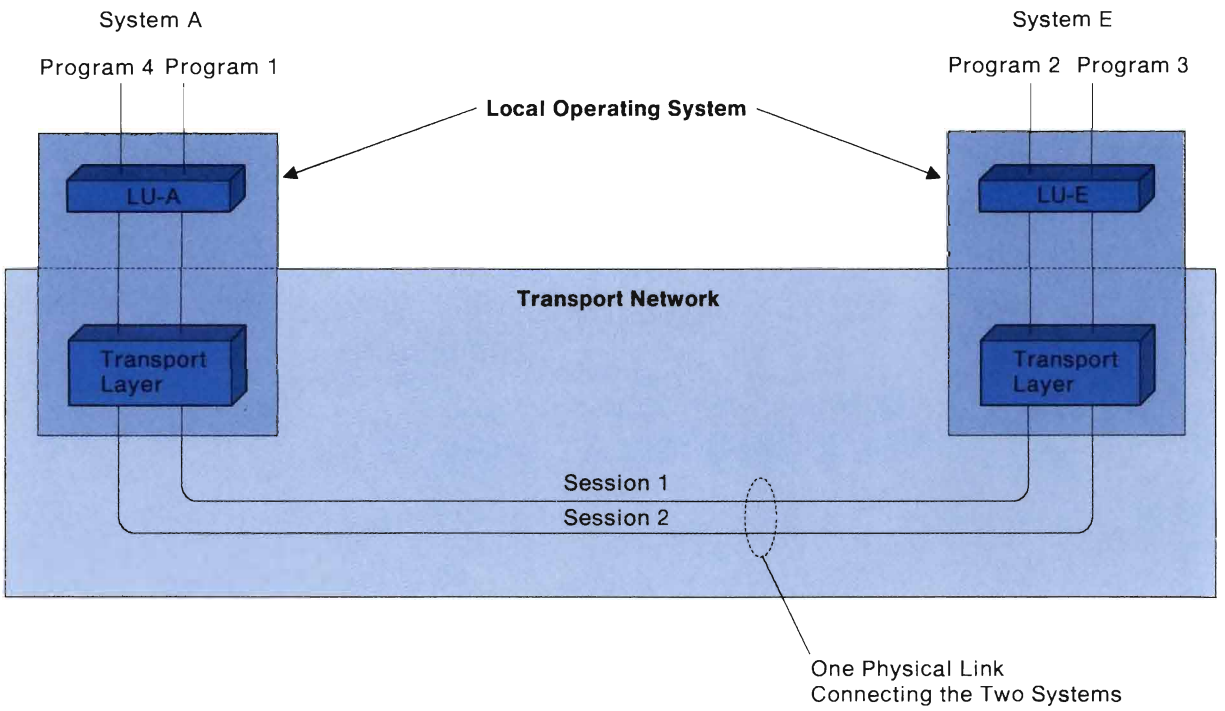


Figure 1 Example of Point-to-Point Communications

communications [3]. The first implementation of this architecture, known as APPC on the System/38, provided the capability to carry parallel LU type 6.2 sessions, thereby allowing multiple partner applications to be active and communicating concurrently. Figure 1 shows Program 1 on System-A using the services of LU-A to establish a conversation with Program 2, which is using the services of LU-E on System-E. Similarly, Program 3 and Program 4 have established a conversation using a different parallel session between LU-A and LU-E.

It can be observed how these distributed applications use the services of the local operating system to communicate on a logical point-to-point, or direct connection, basis. The logical unit (LU) provides the port for an application program to establish conversations and to send and receive data from partner applications. The transport network, which consists of path-control and data-link control elements, is then used to actually deliver the data to the remote LU. In type 2.1 nodes, the transport layer provided data transport on a point-to-point, or one-hop, basis. Therefore, the logical point-to-point connection of LUs and applications was also a physical point-to-point connection of systems, due to the functional capabilities incorporated into the transport network of type 2.1 nodes.

By taking advantage of the layered AS/400 implementation, the path-control layer was enhanced and a set of system tasks was added that resulted in the ability to incorporate advanced functions without affecting the operational characteristics of the applications and the LUs being served. (See the article *The Communications and Networking Structure* for a description of the data communications hardware and software on the AS/400 system.) Figure 2 shows the architectural model of the SNA layers in a type 2.1 node, and how that model was implemented in the AS/400 system. Highlighted is the separation between the LU and the transport network. In this figure, the APPC function manager represents the LU type 6.2 verbs that are issued by the LU, and APPN represents the type 2.1 transport network and the control point functions.

The advanced facilities provided by APPN can be summarized into four main functions, in the order they are automatically performed within the local node:

1. Distributed searches of the network to locate any remote LU requested by a local application.

   This alleviates the requirement to manually define every remote LU with which the local LU may establish a session.

2. Topology and route selection services based on a class of service selected by the user.

   Using the properties of the nodes and links in the network that are maintained in a local topology data base, the best route from the local control point (system) to the remote control point is calculated according to the class of service selected by the user.

3. Activation of a non-configured remote LU.

   Once the correct route is determined, the configuration that was manually configured and activated with APPC is now automatically created and activated by the operating system.

4. Adaptive pacing and transmission priority.

   While establishing the session, the transport layer assigns transmission priority to message units and allocates buffers according to user-specified parameters and systems capacities.

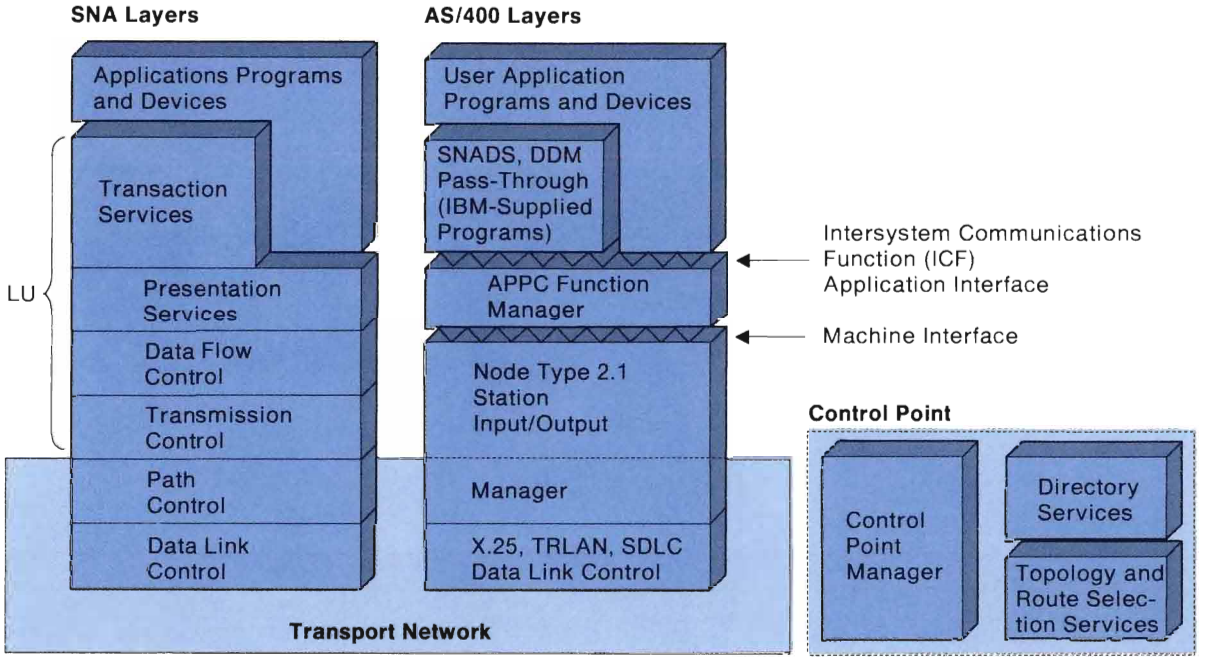


Figure 2 SNA Layers Mapped to AS/400 APPC/APPN Implementation

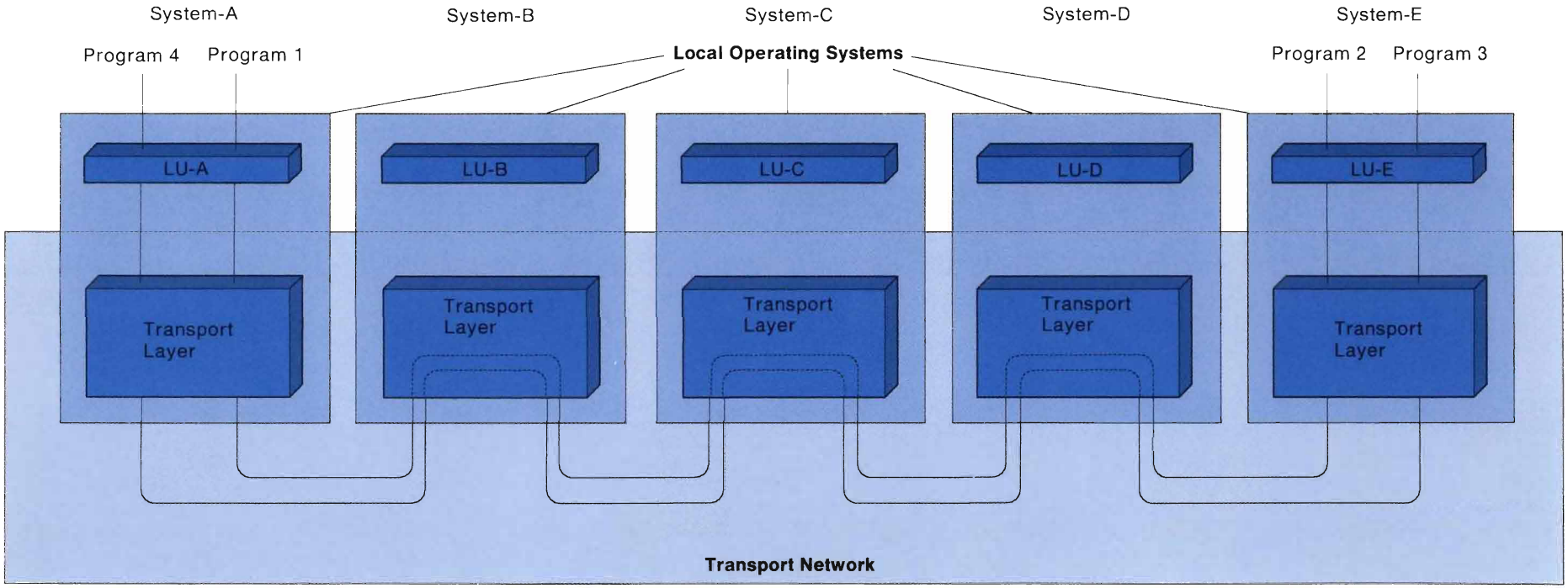
Figure 3, when compared to Figure 1, illustrates how APPC applications and their serving LUs can take advantage of these functions. Shown is LU-A in System-A and LU-E in System-E retaining the appearance of the same logical point-to-point connection as in Figure 1, while the transport network provides for multihop sessions between physically non-adjacent systems.

Planning the Communications Network

The AS/400 system incorporates significant enhancements over the two types of type 2.1 nodes that exist today. Network nodes contain the advanced functions in the path-control layer that allow intermediate routing to be performed within a type 2.1 node. Also included in a network node is a set of tasks, collectively referred to as the control point, that performs the functions of distributed searches of the network to locate a non-configured remote LU and to calculate the best route from origin node to destination node based on user-specified criteria. An end node provides a subset of the network-node function and relies on the services of an attached network node for session requests that involve multiple nodes. End nodes also provide the ability to register their local LUs with a network node server, thereby alleviating the network node operator from having to configure manually the LU names in all of the attached end nodes for which it is providing services.

Figure 4 shows these different node types, connected by the different types of physical media and the related data-link protocols that can be used. Of special interest is the ability for network nodes to route sessions into the wide-area network for nodes that reside on the IBM Token-Ring Network.

All models of the AS/400 system can be configured as either a network node or an end node, and all models may also communicate using synchronous data link control (SDLC) leased and switched connections, X.25 permanent and switched virtual circuits, and the Token-Ring Network. In the sample network, systems A, B, C, D, and E are configured as network nodes that are connected to each other by SDLC leased and switched connections. These network nodes are providing network services for all local users and also for all users of directly connected end nodes. Each system in the network (both network nodes and end nodes) is uniquely identified by a special



RSLL303-2

Figure 3 Physically Non-Adjacent Systems Retaining Logical Point-to-Point Connection with APPN

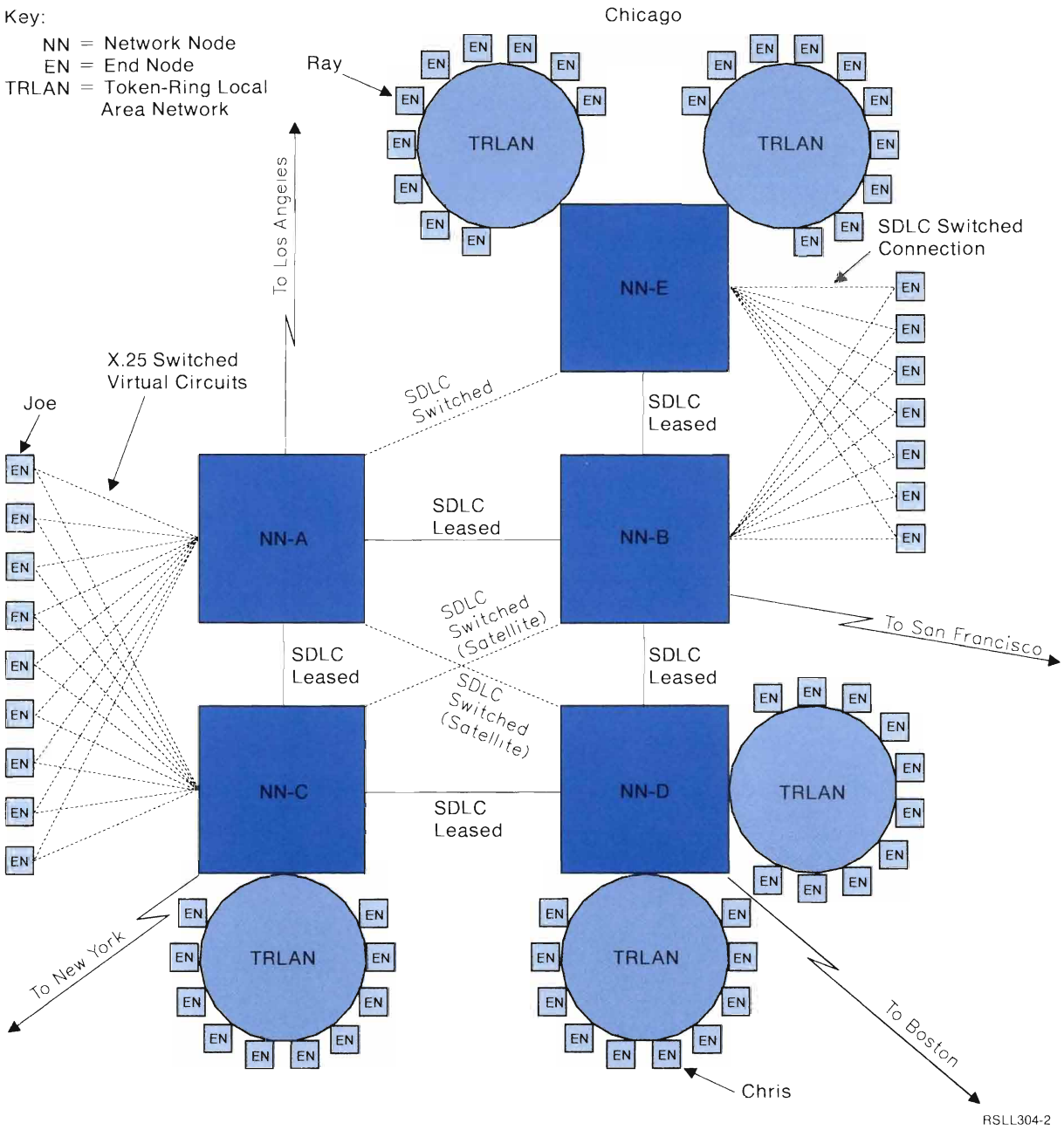


Figure 4 Fully Connected APPN Network

LU name, called the control point name. This name serves a dual purpose: to uniquely identify each node for routing purposes and to be used as an LU name for user applications. Both node types also provide the capability to define additional local LUs within a single node. However, because a control point name uniquely identifies a system in the network, a node can only be defined with one control point name.

The task of configuring an APPN network of any arbitrary size consists of configuring the local control point name and node type, and then the control point name and node type for each adjacent partner. An example would be for network node-A in the sample network shown. First, it is configured as a network node with a local control point name of A. Then, network nodes B, C, D, and E are configured as adjacent network node control points. Finally, all of the end nodes that it wishes served are configured. The characteristics of the links being used are also specified during configuration; default values are provided according to the protocol and physical interface but can be modified by the user.

For the control point to perform the directory services and topology and route selection services, adjacent network nodes (and optionally end nodes) use a pair of parallel sessions, or control-point to control-point sessions, to exchange network information. Management of these sessions is performed automatically by a separate task in each control point and is transparent to the users at these nodes. Token-Ring Networks, X.25 switched virtual circuits, and SDLC switched lines, which are logically switched facilities, can be configured in such a way that they are activated only for user sessions; the connection is dropped when all user sessions have ended. Because the existence of a control-point session will prevent switched facilities from

disconnecting, the planning stage must include decisions about which links will be used for the control points to exchange network information, with the remaining retaining the ability to use the automatic-disconnect feature.

Advanced Functions of APPN

These key requirements drove the APPN design effort: eliminating the need for host processor intervention in peer-to-peer communication; reducing or completely eliminating static definition of network resources; accommodating large networks; and designing for future enhancements. One can view the directory services component as providing a general-purpose search mechanism, and the topology and route selection services component as having the potential to provide routing services over any type of communications medium. In addition, the concept of dynamically creating and activating system objects, which was once done manually, can be extended to include other system objects, and the intermediate routing algorithm that incorporates adaptive pacing and transmission priority can be viewed as another, but not the last, advance in intelligent data transport.

Directory Services

The directory services component in an APPN node is responsible for maintaining locally defined LU names and participating in network searches to help other control points locate requested LUs. Network nodes are also responsible for maintaining the LU names that are contained within adjacent end nodes.

When the transport network receives a session request from an LU it is serving, the services of the control point are employed to provide the necessary routing information. The directory services component is the control point task that must first identify the control point that owns the requested remote LU. Directory services accomplishes this by sending a search request to corresponding directory services components, asking if the requested LU name is known.

As an example, consider the scenario where LU-Chris in End Node-Chris requests a session with LU-Ray in End Node-Ray, as shown in Figure 4. The directory services task in End Node-Chris sends a search request to its network node server, Network Node-D. Network Node-D then searches its local directory data base, and because the requested LU is not found on itself or on any of the end nodes it is serving, it broadcasts search requests to all adjacent network node control points with which it has an active control point session. The search request completes when the search reply is sent from End Node-Ray to Network Node-E and then back to Network Node-D. (Subsequent network searches are optimized by sending searches directly to the control point that positively responded on the previous broadcast search. Also, directory searches are not required when the remote LU name is equal to the name of a network-node control point.)

As a second example, consider the case where End Node-Chris requests a session with End Node-Joe. The search request is sent as before, except that End Node-Joe has elected not to establish control point sessions with any network node that would prevent using automatic disconnect over the X.25 switched facilities. Therefore, Network Node-C and Network Node-A will not forward the search request to End Node-Joe, but will return positive responses and supply the necessary information about the link to End Node-Joe.

In both of these examples, the directory services component at Network Node-D has obtained the necessary information: identifying the control point that owns the requested LU, along with obtaining information about the links that connect the end nodes to the intermediate routing portion of the network. This is how the distributed directory search function is central in alleviating the user from manually configuring remote LUs. In addition, because the distributed search function is present in all network nodes, the reliability of this function is enhanced, as no single point of failure exists within the network.

Topology and Route Selection Services

The topology and route selection services component uses the control point sessions between network nodes to exchange information and build a topology data base. This topology data base includes all network nodes, links between network nodes, and associated characteristics of these nodes and links, so every network node can maintain a fully replicated copy of the intermediate routing portion of the network. This topology data base in network nodes is kept current using updates that are transmitted throughout the intermediate routing portion of the network whenever a new resource (node or link) is activated or the characteristics of an existing resource change.

At session request time, the user supplies the type of service being requested by specifying a mode name. This mode name is associated with a class-of-service definition that is used to determine the most desirable route between the origin and destination control points. The class-of-service definitions specify the characteristics that nodes and links must possess to be included in the route selected for the user. This allows the route selection algorithm to determine first if a node or link is acceptable, and from the set that is acceptable, to calculate the best route dynamically.

Because class-of-service definitions may vary, different sessions may use different routes between the same origin and destination control points, depending on the mode name and associated class of service selected. With the

AS/400 system, five mode and class-of-service definitions are automatically created during the initial program load (IPL). These definitions allow users to choose between routes and transmission priorities that are favorable for batch or interactive traffic. Users can also modify these supplied definitions or create their own class-of-service definitions to control session routing according to their requirements.

After the session origin and destination control point names have been resolved by the directory services component, the topology and route selection services component uses information that it has stored in its local topology data base, and any information possibly returned on the search reply (when end nodes are involved), to calculate the best route from the origin control point to the destination control point. Because topology data base updates are sent and received by the topology and route selection services component as characteristics of any resource change, every route is calculated with the most current information.

Consider when LU-Chris is attempting to establish an interactive session with LU-Ray. The selected class of service for an interactive job specifies that the links with the fastest line speed and shortest propagation delay are preferred over links with a slower line speed and longer delay. Assuming in this example that all the nodes and links were operational and available for use, the route End Node-Chris to Network Node-D to Network Node-B to Network Node-E to End Node-Ray would be calculated.

As a second example, consider the scenario where LU-Chris is requesting a session with LU-Ray for a batch job. The class of service for batch jobs prefers a route with a longer propagation delay in an effort to leave the shorter propagation delay links for the interactive jobs. This time, the session route calculated would be End Node-Chris to Network Node-D to Network Node-A to Network Node-E to End Node-Ray. The route calculated would cause the switched links connecting the network nodes to be activated for this session, because the topology and route selection services component calculated that activating the switched satellite link with a fast line speed, but long propagation delay, yielded the best route for the batch class of service.

Dynamic LU Activation and Session Establishment

Once the route has been determined by the topology and route selection services component, APPN automatically creates and activates the LU description associated with that path. This is the same LU description that would have been created if the user had manually configured each LU for each transport link. By dynamically creating the description of the LU, the network eliminates the need for explicit system operator definition for each remote LU to which the local system communicates. (For more information, see the article *A Structured Approach to Data Management*.)

To establish a session between the local and remote LUs, a session activation request is routed through the transport network. The session activation request contains an ordered list of the nodes and the links used to reach the destination LU. As the activation request crosses the network, each intermediate node puts in place a temporary routing entry. The routing entry contains addressing information, generated at the previous node, for use on the link that the activation request arrived on. It is then automatically assigned a second address for use on the outgoing link. This allows subsequent session traffic to be routed simply by giving the session address of the origin LU, and eliminates the need for fixed routing entries in the transport network (see Figure 5).

Adaptive Pacing and Transmission Priority

The objectives for the transport network were to provide efficient and equitable data transfer for all sessions, while still allowing selected sessions to be assigned priority. APPN accomplishes this through the use of adaptive pacing and three levels of transmission priority available for user sessions. Note that one transmission priority, network, is available only for network control functions.

Adaptive pacing allows the receiving transport layer to change or adapt the pacing window size based on its buffer resources and traffic patterns in the network. The previous APPC flow-control algorithms depended on fixed pacing. The pacing window, or the number of message units that could be transferred over a session before receiving an acknowledgment from the receiving transport layer, was negotiated at session establishment and was fixed for the duration of the session. The receiving transport layer can now allocate its session buffers dynamically, efficiently using its available resources. It also has the ability to slow down the transfer of data, or even stop receiving at any node of any session, thereby maximizing equity in the transport network by adjusting the flow of messages for any session that may be contributing to congestion problems in the network.

The transport layer also allows message units to be transferred through the network at different priorities. Before APPN, the type 2.1 transport layer would simply transmit message units on a first-in, first-out basis. There was no way of specifying or allowing a particular session's message units priority transmission over the message units for any other session. This allowed batch-like applications to consume the available transmission media bandwidth much more readily than applications that were interactive in nature, or had short bursts of data to transfer. APPN allows

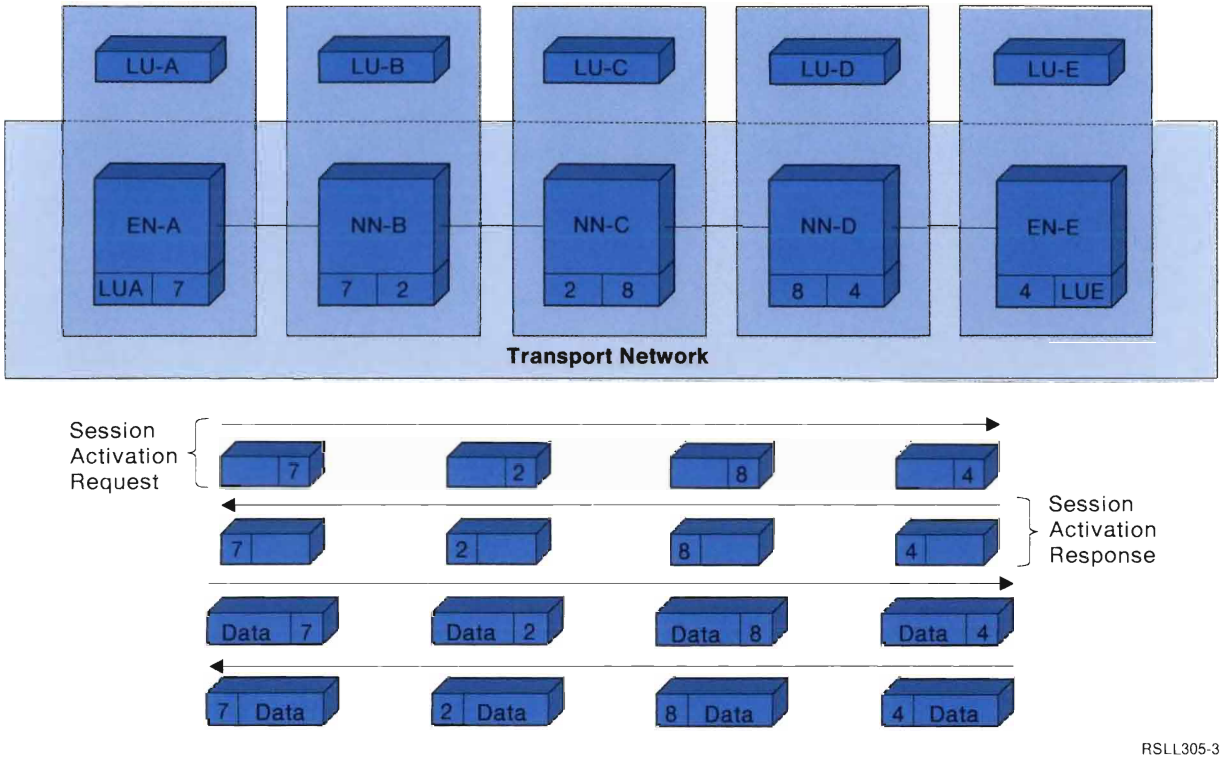


Figure 5 Single Route Activation and Data Transfer

the user to configure three session-level priorities: high, medium, and low. The transmission priority is carried in the session activation request at session establishment, allowing the two halves of the session and each routing entry along the session path to store the same transmission priority.

To ensure that lower-priority message units are not preempted indefinitely by higher-priority message units, an aging mechanism was developed. The aging mechanism consists of a service number, a transmission priority number assigned to each transmission priority, a scheduling queue, and a key-ordered priority queue. The transmission priority numbers provide a **priority factor** for each transmission priority. The following values were assigned for each transmission priority:

- Network = 0
- High = 8
- Medium = 16
- Low = 32

The service number is initialized to zero and is incremented each time a session control block is serviced. This number enforces first-come, first-serve scheduling for a given priority, and also provides an aging factor for unequal priorities. Special wrap logic is also supported to manage the path control priorities when the service number wraps.

The scheduling queue contains a set of session control blocks that represent each half session or routing entry that has pending message units to transmit. The session priority (high, medium, or low) is stored within each session control block. The message units available for transmission are attached to each of their session control blocks. The priority queue is ordered by ascending key and contains one element for each of the session control blocks currently on the scheduling queue. The keys of the elements on the priority queue are the sum of the service number and the session control block's transmission-priority number.

The transport layer always dequeues the first element on the priority queue to service a session control block. It then transmits as many message units as it possibly can over the underlying link and then increments the service number. The key of the priority queue element (which is the sum of the service number and transmission priority number) is then modified, and the element is enqueued to the priority queue *before* the first element of greater value. This allows the priority for sessions to decrease gradually while still enforcing the first-in, first-out ordering for sessions of equal priority.

Figure 6 shows an example of several message units being received by path control to transmit the effect on the priority-queue elements keys, and the order of transmission.

Figure 7 shows the relationship between the scheduler and priority queues.

Implications for APPC/LU Type 6.2 Applications

One of the design objectives of SNA, carried out in the implementation of AS/400 APPN, was to allow resources, such as application programs and data files, to be relocated without affecting the remote applications that access them. This allows transaction service-layer programs and user

Message Order	Service_Number	Trans_Priority_#	Priority_Queue Key
1	0	Low = 32	32
2	1	Low = 32	33
3	2	Med = 16	18
4	3	High = 8	11
5	4	High = 8	12
6	5	Low = 32	37
7	6	Med = 16	22
8	7	Low = 32	39
9	8	Network = 0	8

The actual order of transmission is message: 9, 4, 5, 3, 7, 1, 2, 6, 8.

RSLL307-2

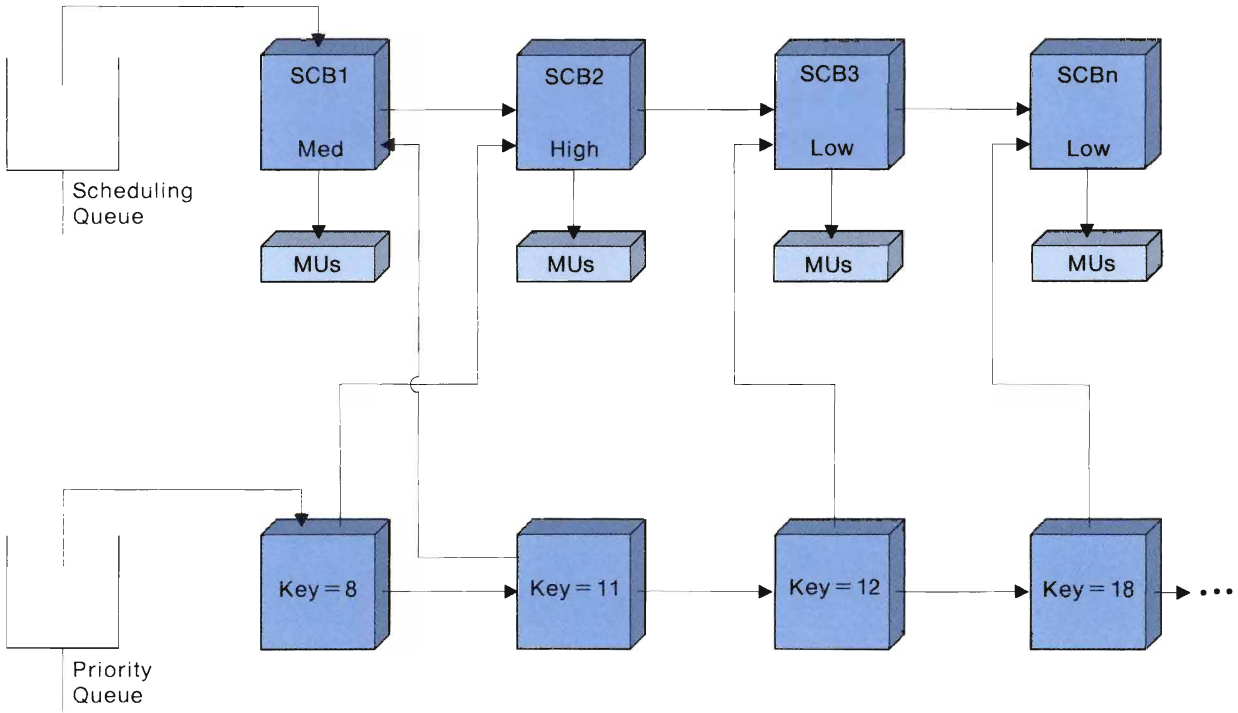
Figure 6 Transmission Priority Example

application programs to refer to a resource by its name without knowing the actual address of that resource or the configuration of the network.

The ability for a single control point to configure multiple local LU names provides the vehicle to move resources associated with a certain LU name without affecting the more permanent control point name. This is made possible by the directory services function of determining the owning control point for an LU name, and then by the topology and route selection services function of calculating the path between the origin and destination control points.

Consider the scenario where applications (shown in Figure 4) access a user data file named USERINFO that resides on Network Node-E that is associated with an LU name of USERINFO on Network Node-E. During the course of normal operations, it is determined that Network Node-D would be a more appropriate system for this file. Using an IBM-supplied transaction-level program called object distribution, Network Node-E would send the file to Network Node-D. Network Node-E would then delete the local LU name of USERINFO, and, at the same time, Network Node-D would add USERINFO as a local LU name. These steps allow remote applications to continue to access the file USERINFO associated with the LU name of USERINFO. The ability of the control point tasks to recognize that the LU name of USERINFO now resides in Network Node-D, and the ability of the transport network to provide the routing transparently, is key in shielding users and LUs from the real address of a resource.

Using an IBM-supplied application called display station pass-through, which allows for remote-system sign on, the sequence above could be performed from a single control station. Therefore, the required involvement of users on each system can be minimized, especially if skilled network management personnel are centrally located.



RSLL308-2

Figure 7 Relationship between Scheduling and Priority Queues

Conclusions

AS/400 advanced peer-to-peer networking builds upon the non-hierarchical, point-to-point Low-Entry Networking protocols implemented in type 2.1 nodes. Advanced functions developed for the AS/400 system free users from the detailed manual tasks that were required with previous networking solutions.

These advanced functions are: distributed directory searches; topology and route selection services; dynamic logical unit activation and session establishment; and adaptive pacing and transmission priority. Distributed directory searches provide the current address of a remote LU for user applications that only know the LU by name. The topology and route selection services component selects the best nodes and links to use based on a set of user-specified criteria to access the remote LU. Dynamic logical unit activation and session establishment serves as a placeholder for current communications. Adaptive pacing and transmission priority allows the transport network to adjust the flow of session traffic.

The layered structure of the operating system allows new and old products to coexist gracefully, and additional functions to be added in a natural manner to meet future requirements. This is highlighted by the enhancements made to the transport network while preserving the APPC application program interface. Advanced peer-to-peer networking demonstrates the AS/400 commitment to provide the best networking functions in the industry.

References

1. Baratz, A. E. et al, *SNA Networks of Small Systems*, **IEEE Journal on Selected Areas in Communications**, SAC-3, Number 3, 416-426. May 1985.
2. Sundstrom, R. J. et al, *SNA: Current Requirements and Direction*, **IBM Systems Journal**, Volume 26, Number 1, 13-36. 1987.
3. Gray, J. P. et al, *Advanced Program-to-Program Communication in SNA*, **IBM Systems Journal**, Volume 22, Number 4, 298-318. 1983.

A Structured Approach to Data Management

Highlights the advances in display and communications data management, describing the data management structure necessary to support them.

Carol A. Egan and Daniel S. Brossoit

Introduction

The AS/400™ data management structure greatly simplifies the process of accessing data from different media by providing a consistent system-wide method of data definition and access. AS/400 data management supports data base, an extensive list of devices, and communications capabilities to many different systems. This structure also provides the underlying support for application portability by providing a common interface for applications running in Operating System/400™ (OS/400™), in the OS/400 System/36 environment, or in the OS/400 System/38 environment.

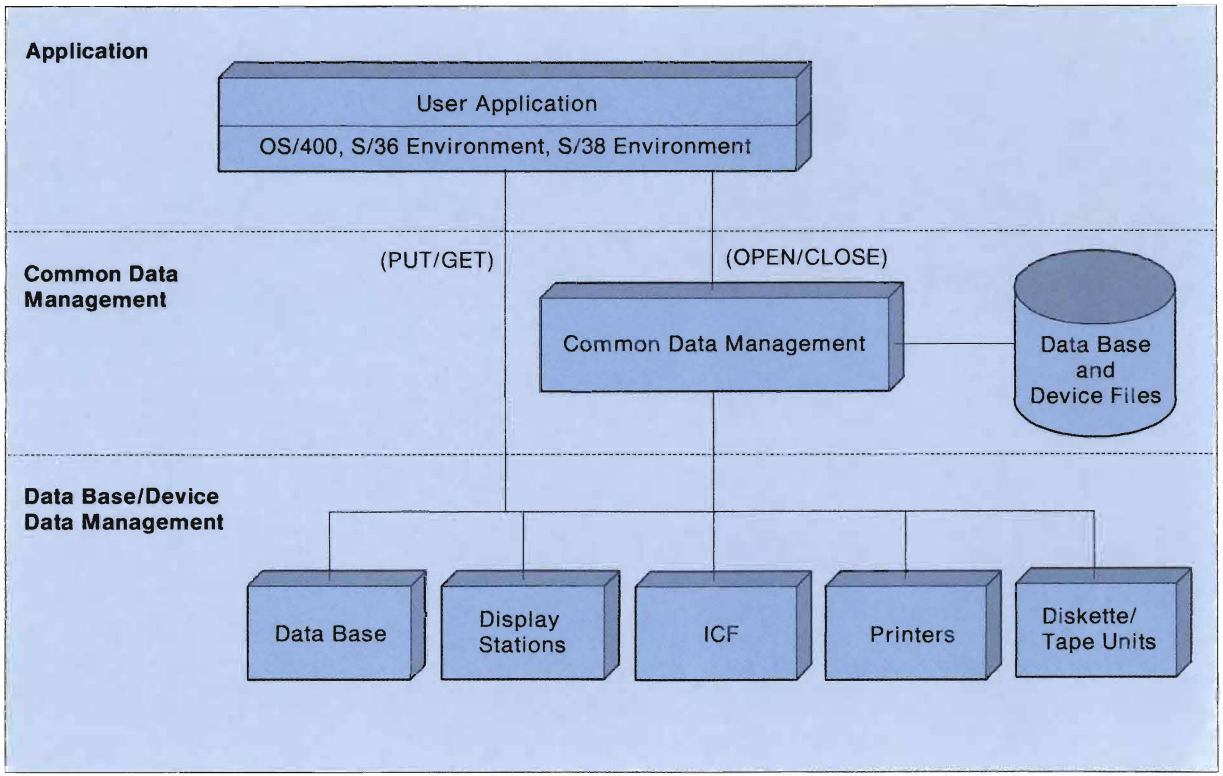
File processing and externally described data have been implemented as the interface to the AS/400 data management function. The data management structure has been integrated in the AS/400 system to incorporate this interface. Display and communications structures, with the primary focus given to Intersystem Communications Function (ICF) data management, provide a significant advancement in the AS/400 data management structure.

File Processing

The file-processing interface serves as a basis for AS/400 data management support. A file is the object used to access data. Files supported include data base files and device files. Data base files provide access to the data base, while device files provide access to input/output (I/O) devices, such as display stations, printers, and remote systems.

All file types support the following base set of file operations:

- OPEN (create the path for data transfer)
- PUT (send data to a device or data base)
- GET (retrieve data from an input device or data base)
- CLOSE (remove the path for data transfer)



RSLL312-4

Figure 1 Data Management Structure

The data management file support is structured with two distinct divisions: common data management and data base/device data management. Common data management provides functions needed by all data management support facilities. This includes the open and close interface for all device and data base files. Data base/device data management is subdivided into multiple structures to provide the functions unique to the various devices supported. A remote system is treated as another device on the system and is supported through the ICF structure. Figure 1 illustrates the relationship of the various data management functions.

The OS/400 file interface allows the ability to have externally described data. This support is provided by data description specifications (DDS), which are part of the file description. Externally described data allows for centralization of data definitions in the file external from individual programs. Using it, programmers take full advantage of the system's data management, improving their productivity as well as program and data integrity. (Refer to *Application Development Support* for more information on the programmer productivity provided by the AS/400 system.)

Common Data Management

Common data management provides the foundation for file processing on the AS/400 system. The file is identified and the relationship between the file and the program is established with the use of a file OPEN interface, by which common data management creates an open data path. The open data path provides the link between the program and the different file-specific routines of the underlying data management. In addition to providing the link to the processing routines, the open data path also contains all the file-status information needed by the application to access the file. The application has access to the open data path through a user-file control block. The user-file control block, which is created and maintained by the compiler on behalf of the application, is a consistent link to any file, regardless of the file type. Figure 2 shows this link between the program, file, and open data path.

OPEN and CLOSE operations are processed through common data management. PUT and GET operations, which use the open data path and user-file control block interface, are routed directly to the appropriate data base or device data management routine. The appropriate file processing routine is called to process the operation due to the link established during OPEN. This ability to tie I/O operations to specialized file processing routines during the OPEN operation provides the flexibility for using the same open data path and user-file control block interface to process operations and data to distinctly different media, such as data base and display stations. An application program accesses the different media by opening two different files, which creates two separate open data paths linked to the appropriate processing routines. Issuing PUT and GET operations to each file transfers the data to the corresponding media.

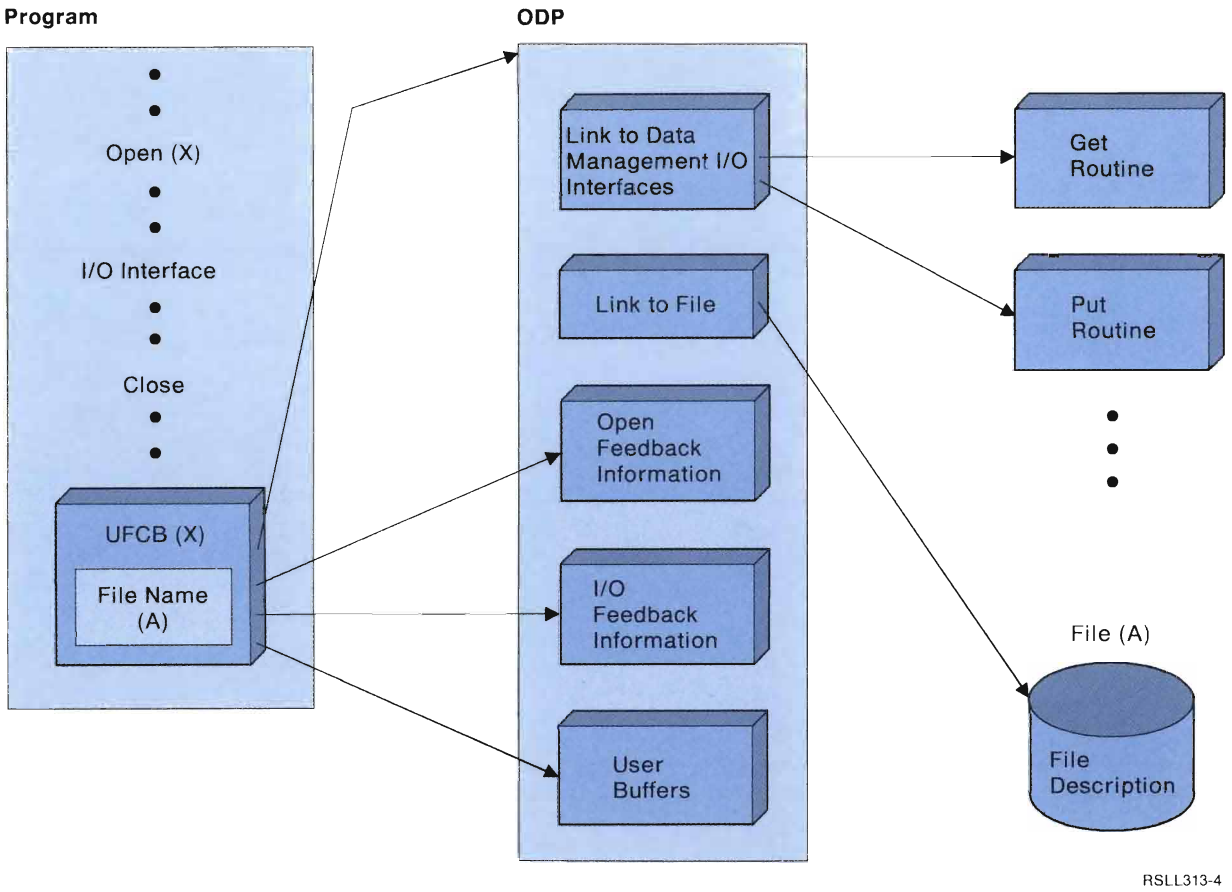


Figure 2 Opened File Structure

Data Base/Device Data Management
The data base/device data management routines provide the specific support for data base and each of the AS/400 devices. Advances in data base, display station, and ICF data management are some of the key advances in AS/400 data management. (For advances made in data base, see the article *An Integrated Data Base* .)

Display Data Management
Display data management, part of the AS/400 data management structure, provides the application interface for display stations. The application issues I/O requests through the high-level language read and write file operations. The display characteristics are defined in the file with DDS keywords. Because the underlying display data management structure converts the application I/O requests to the appropriate device control information, the application is not dependent on the type of display station being used. Although display stations differ in function and can be locally or remotely attached, the single interface lets any given application program work with any display station.

The application interface can be expanded using DDS keywords, so new function can be added easily. An example is application help; this function is provided to the application, in a manner consistent with the rest of the display station interface, by DDS keywords such as Help Area (HLPARA), Help Record (HLPRCD), and Help Sequencing (HLPSEQ).

ICF Data Management
Comparable to the display data management function, ICF data management provides the AS/400 system's single interface to communications. The ICF interface supports a full range of function, while still maintaining a file interface that is consistent with data base and all other device support. To make communications a logical extension to the file interface, the ICF interface required special considerations, such as the ability to request that a remote process be started, to support both interactive and batch remote communications.

Consistent Interface. The basic concept of ICF is to isolate applications from the complexities of communications protocols and hardware. The underlying AS/400 communications structure handles the protocol and hardware characteristics for the application program.

Communications functions are grouped into communications types and integrated into the communications structure as system routines, called function managers, below ICF data management. ICF data management handles the file operations and data for the application. The function manager handles the communications protocol needed to perform an operation. Each communications type is designed to work with a group of remote systems and hardware devices through a specific communications method, such as binary synchronous communications (BSC) or Systems Network Architecture (SNA). The communications types supported are advanced program-to-program communications (APPC), SNA upline facility (SNUF), BSC equivalence link (BSCEL), and asynchronous communications. Figure 3 shows the relationship of the various data management functions.

ICF functions include:

- Establishing a communications session between the local system and a remote system.
- Starting a process on a remote system. (The process on the remote system can be a job. This allows the local application to start a job on the remote system without operator intervention.)
- Sending and receiving data.
- Ending communications with a remote process.
- Ending a communications session.

The ICF interface is designed to support interactive communications. Thus, an application can be written to perform a batch transfer of data, or to send a request for a single data record to a remote system and then wait for the reply to be received. To facilitate interaction, the ICF interface provides the ability to start processes on remote systems, and allows remote systems to start jobs on the local AS/400 system.

The ability to provide a consistent interface across all communications types has been achieved by mapping specific communications functions into DDS processing keywords. The underlying support interprets these generalized DDS keywords in terms of specific communications protocols. A base set of these keywords is supported by all communications types to provide equivalent base-level support. For example, every communications type supports starting a process on a remote system with the EVOKE keyword. Additional DDS keywords are communications-type specific to allow full use of the communications protocol.

From an application perspective, ICF merges the best characteristics of the System/36 and System/38 communications interfaces. For instance, ICF supports externally described data and DDS keywords, a concept from the System/38 interface. ICF also supports system-supplied formats, compatible with the System/36 System Support Program Interactive Communications Feature (SSP-ICF) operations, which use program-described data and provide similar functions of DDS keywords. ICF functions are a superset of the functions provided by the System/38 communications DDS keywords, System/36

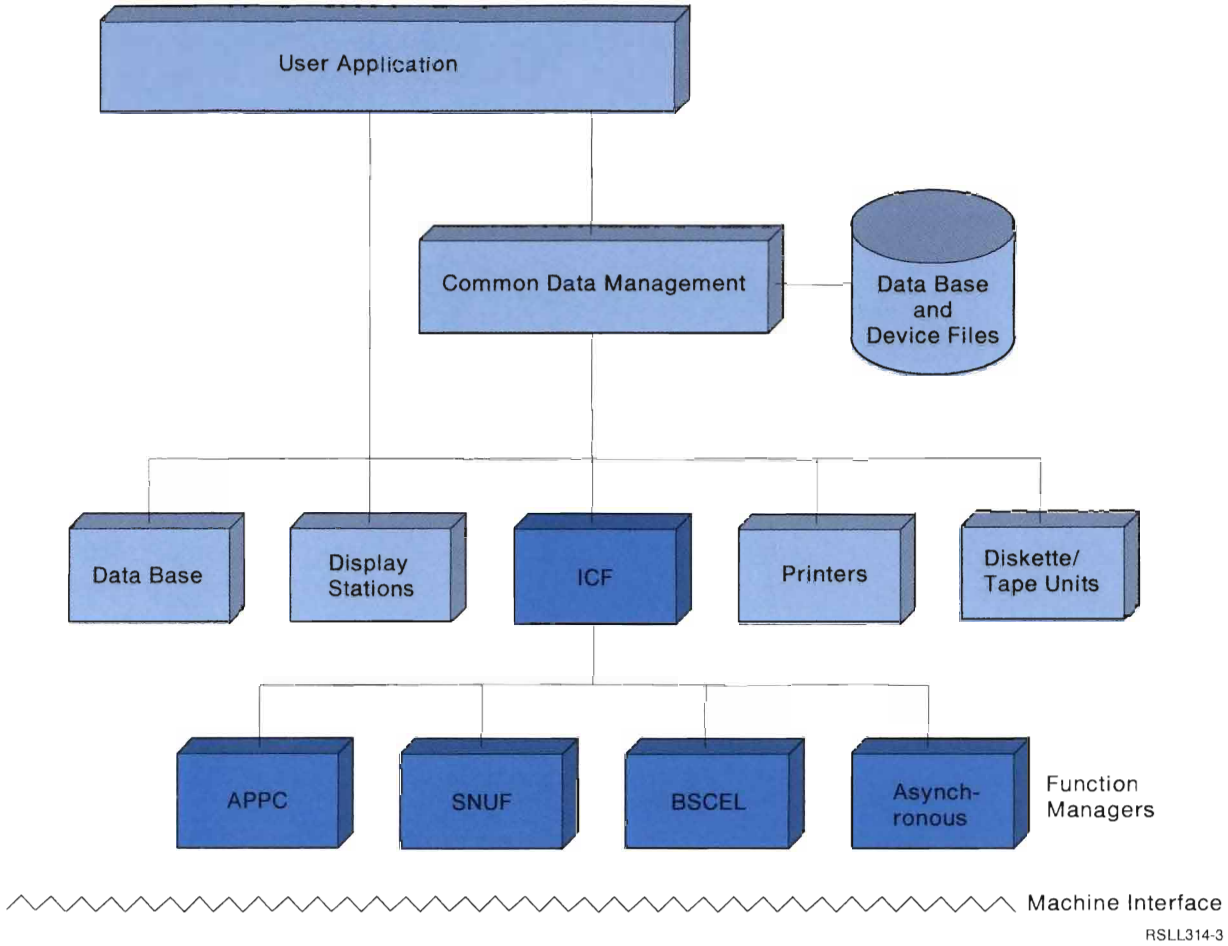


Figure 3 Communications Data Management Structure

SSP-ICF operations, and System/36 interactive data definition utility (IDDU) support for communications.

Remote Resource Independence. An application program maintains independence from a specific remote resource, such as an APPC logical unit (LU) or an asynchronous communications display station, through the use of a program device. All operations in the application program are issued to a program device name instead of to a specific remote resource. Because of this use of a program device name, there is no specification within the high-level language program to a particular remote resource. Because this association is removed from the application program, the program can communicate with various remote resources without modification.

The program device is directed to a remote resource through the use of a remote location name. The program device and the remote location name are bound by defining a program device entry with a control language (CL) command before starting the program. OS/400 support provides both an early binding capability of program device and remote location name through the use of the Add ICF Device Entry command, and a late binding capability through the use of the Override ICF Device Entry command.

The remote resource is represented by a set of one or more device descriptions that contains the same remote location name. While program device names allow application programs to be independent from specific remote resources, remote location names allow program devices to be independent from specific device descriptions. Remote location names allow a single logical name to be used to access generically a set of one or more device descriptions. The program device is bound to a specific device description at session allocation time. All operations at the machine interface are issued to a specific device description (see Figure 4).

Additional function is provided using a remote location name to gain access to any device description that contains the same remote location name. These functions include:

- Selecting a route through the network at session allocation time and then automatically creating a device description that reflects the route selected based on the remote location name requested. This support is provided by the networking capabilities of advanced peer-to-peer networking (APPN) that are accessed through the APPC communications-type interface. (See the article *Advanced Peer-to-Peer Networking*.)

- Allowing a single program device to represent multiple device descriptions.

The mapping from remote location name to device description is communications-type dependent.

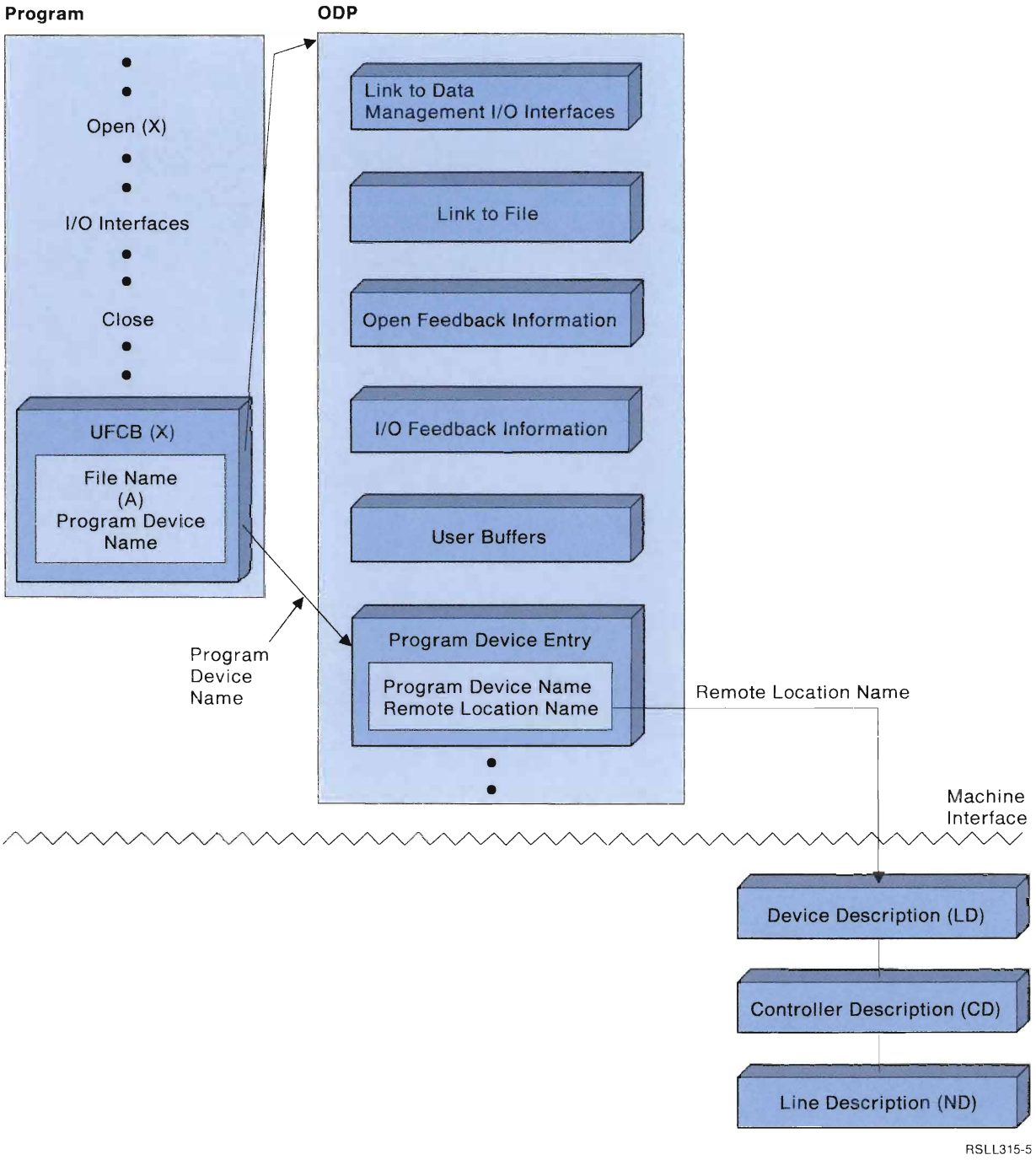


Figure 4 Remote Location Name Correlation

For the asynchronous communications and BSCEL communications types, a one-to-one mapping of remote location name to device description exists. Binding a program device name to a device description involves selecting the device description that contains the remote location name specified in the program device. For APPC and SNUF, a one-to-many relationship between the remote location name and device descriptions can exist. For the SNUF communications type, the system uses the first available device description, while for the APPC communications type, the system selects the device description that reflects the route the session is taking through the network.

Because the system dynamically maps a remote location name to a specific device description, the application program is also dynamically mapped to a specific communications type at session allocation time. An application program is also given the ability to pre-select a particular device description (and communications type) by specifying a device description name in its program device entry definition.

Multiple Environment Support

Because of the flexibility of the data management structure, the same internal interface can be used between the user application and the underlying data management structure, regardless of the file type, language, or environment being used. Therefore, the same data management structure can support applications running in OS/400, in the System/36 environment, or in the System/38 environment.

A consistent data management structure does not restrict the ability to portray different application interfaces. Two methods are used by data management to determine which interface to use. The first is by defining DDS keywords that allow the application to indicate the characteristic desired. For example, when System/36

applications are migrated to the AS/400 system, a display file is created with a DDS keyword that will cause the display to be automatically cleared on all output operations. System/38 environment applications do not support this keyword, and consequently the display is only cleared on the first operation to the display. An OS/400 file can be created with or without the keyword, therefore an OS/400 application can choose either interface.

A second method used to determine the interface is to define the interface characteristics based on the type of application and the environment it is running in. For example, System/36 multiple requester terminal (MRT) applications are supported for both display stations and ICF in the System/36 environment. Also, read under format is supported for System/36 display station and communications applications in the System/36 environment. These functions are provided by the System/36 environment and data management extensions. (See the article *The System/36 Environment* for more information.)

Conclusions

ICF data management is a significant advance in the AS/400 data management structure. ICF provides a consistent, easy-to-use interface across various communications types that isolates the application from the complexities of communications protocols and hardware. It also allows the ability for the application to select the remote resource with which it is communicating without changing the application program.

To meet the immediate needs of migration from the existing systems, the same data management structure provides support for Operating System/400 and the System/36 and System/38 environments, which helps provide application portability from a System/36 or a System/38.

The AS/400 data management structure, and the file processing interface it supports, provides a consistent system-wide method of managing data across different media. The structure was designed to provide easy expansion of function for applications of the future.

Integrated Office Support

Describes how AS/400 Office employs the capabilities of hardware and software products to make office tasks simple and efficient.

David G. Wenz, Richard J. Lindner, James H. Bainbridge, Stephen J. Cyr, Barry W. Hansen, and David N. Youngers

Introduction

The major objective of AS/400™ Office is to improve office productivity. To accomplish this, it must efficiently integrate the set of office functions available to office workers. The system must provide facilities that allow the user to organize and control information assets in a manner comfortable to the user. It must allow office workers to communicate easily, quickly, and comfortably, much like a phone conversation or face-to-face meeting.

This can be difficult when workers throughout a company use several different tools in their day-to-day activities. Multiple systems located at different sites further complicate the problem of communication. AS/400 Office solves these problems, allowing communications to flow easily from user to user on one or more IBM systems.

The solutions provided by AS/400 Office can be described in terms of its major elements. The filing system provides underlying support necessary to integrate the hardware and software product capabilities into the efficient office required for today's businesses. Electronic mail allows anything in the filing system to be sent as easily as mailing paper today, but much more efficiently. The efficient use of personal computers is accomplished by applying cooperative processing techniques. A flexible and powerful editor transparently integrates the system functions into the processing of office tasks. And, the AS/400 Office menu makes all office functions accessible, providing a user-friendly environment for all levels of expertise.

The Filing System

The filing system is the heart of the AS/400 integrated office environment. It is a *single container* for all office objects that can contain the data for any office product being used. Figure 1 shows the various types of data that can reside within the filing system. Mail, documents, programs, and files are among the traditional objects that can reside in this filing system, but it can also contain spreadsheets, images and graphs, personal computer (PC) programs, and PC files. Data can be shared among users, with authorization controls specified by the owner of the data. (The article *Security* describes the authorization capabilities in more detail.)

Document organization and control is a key element of office work. The AS/400 Office filing system provides document library services that allow a user to handle these tasks in a comfortable manner, using the filing system as an electronic filing cabinet complete with folders. Folder management services allow the user to organize office objects using these folders. Folders can contain other folders, and can be interactively searched for an office object. Or, familiar search procedures can be used to get a list of documents conforming to specified selection criteria.

The key to the filing system capabilities is the design of the document. The text of each document is stored as a separate system object allocated from single-level storage. Associated with the text of each document is another portion, called the attributes, which includes items such as subject, author, and other keywords that may be used to search for or identify a document. The attributes portion of the document is stored within the system's integrated data base, which provides powerful query search capabilities. (For more information about the functions and capabilities of the system's data base support, see the article entitled *An Integrated Data Base*.)

All documents in the filing system reside in the document library. This library conforms to the IBM strategic Document Interchange Architecture (DIA) [1,2]. Document content, including format and structure, is also governed by a strategic architecture, the Document Content Architecture . AS/400 Office conforms to Level-2, for final form (print format) documents [3], and Level-3, for editable documents, which can contain image or graphics [4]. DIA is made up of three components: library services, remote library services, and distribution services. The library services component can search, store, and retrieve documents in the local DIA library. The remote library services component can store and retrieve documents in a DIA library on another system within the network. If the document is on another AS/400 system, it can also be checked out for editing and checked in when complete. This allows shared processing without the danger of work being destroyed by another edit session. The distribution services component allows the

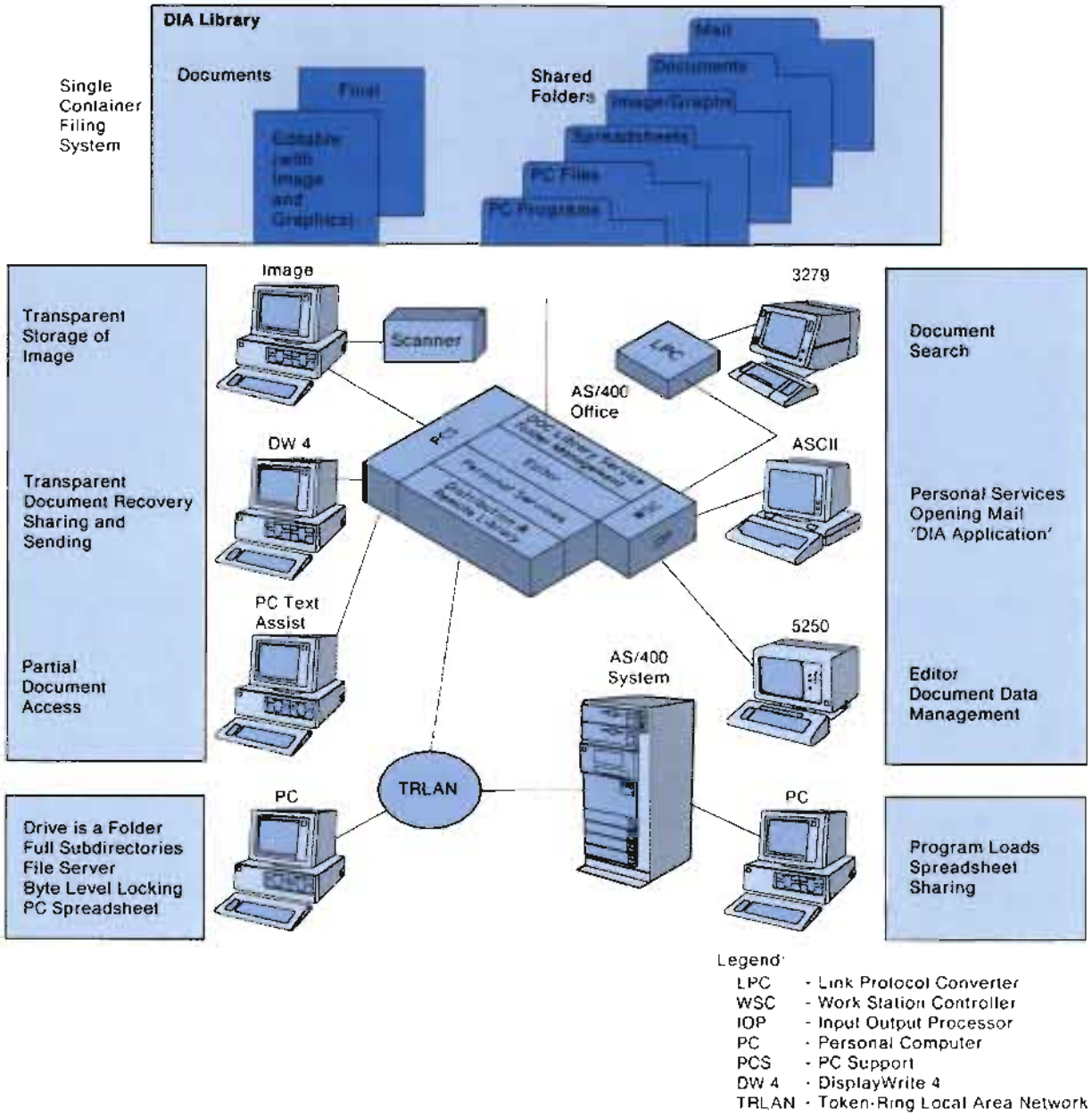


Figure 1 AS/400 Office Filing System Features

user to list, receive, and cancel mail from the mail log, and distribute documents and messages. Folder management services ensure the internal documents are converted to interchange format before they are sent into the network.

Folders can be very useful for organizing documents. However, some documents do not require a folder, and can be placed directly into the DIA library outside of a folder. Users may also subset the documents in the library using the search function to create document lists. The list is a separate system object that is stored in the library. It may contain documents from inside and outside folders within the library. Figure 2 shows the AS/400 document library with folders.

Another aspect of the filing system is the two interfaces for work stations. The work station controller interface supports a variety of dependent display stations. The support is tailored to the capability of the display station. The AS/400 PC Support interface supports the use of attached personal computers. PC Support services requests using the filing system to make the PC files in host folders appear as though they were in the PC storage. Files residing in the storage of other AS/400 systems in the network can also be accessed by either type of work station. The distributed library services of the filing system handles these requests using AS/400 communications support.

AS/400 Office provides a significant advantage for PC users by providing transparent access to PC files on the host system. This means PC applications can run using shared files from the host system. To achieve this transparent processing, the PC naming convention, complete with qualifiers, is supported within the document and folder naming conventions of the AS/400 Office filing system. The filing system does all

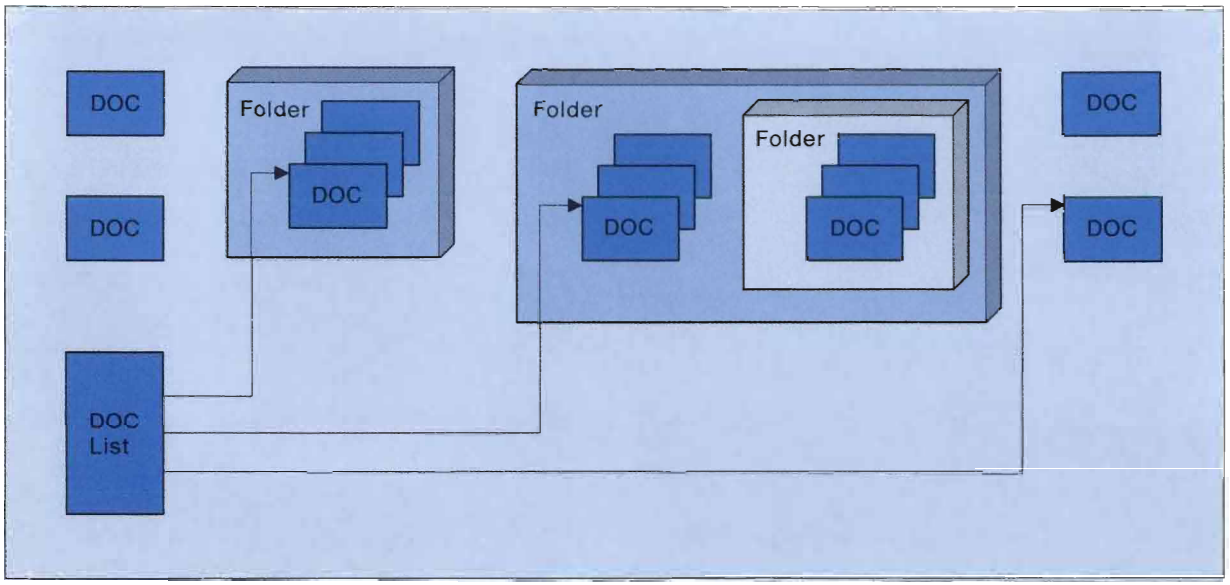


Figure 2 AS/400 Document Library Objects

appropriate locking and sharing automatically and provides recovery services. If a session is ended abnormally for any reason, the user has an option to retain or discard any changes made to the file.

Electronic Office Mail

Electronic office mail is an important element of the comfort and productivity associated with AS/400 Office. The ability to communicate with other users on this or other systems, with path transparency over various communications protocols, sets this offering apart. Figure 3 shows AS/400 Office with a variety of display stations able to communicate locally using Office. It also shows the integrated support that makes remote mail functions easy to use.

AS/400 Office services are an integral part of Operating System/400™ (OS/400™) and therefore can sheild the user from the complexities of handling communications. Menu options allow the office worker to display the items in a folder or to compose a note using a simplified note editor. A simple selection or single command can send an item or note to another user or list of users. The system automatically handles the distribution based on the system distribution directory, from which the system determines the location of recipients. Mail directed to a user on a remote system, who is not explicitly defined in the local system's directory, is automatically handled by the system. This allows one system with a central directory to be used as a mail router, with directory maintenance consolidated in only one place.

Maintenance of distribution lists is also simplified by AS/400 Office. When a user description is changed or removed in the system distribution directory, all locally defined distribution lists are automatically updated. Distribution lists can also be easily tailored by the user when sending mail. The user can expand a locally defined list and, optionally, add or remove entries for the mail being sent. This ability to tailor distribution lists reduces the number of lists needed, and thereby reduces the maintenance required. The number of lists can also be minimized using the system's ability to send mail to a distribution list defined on a remote system. The remote system expands the list and directs the mail based on the content of the list.

Office users are informed of all new mail by a highlighted message on the main office menu. They can also be informed of high priority mail, when not at the main menu, through the system message support. When priority office mail is received, a message is sent to the user's system message queue to tell the user about the arrival of the mail. The message is shown to the user even while system applications other than office are running.

Systems Network Architecture distribution services (SNADS) provides distribution and confirmation of delivery for mail sent to users on another AS/400 system, Distributed Office Support System/370 (DISOSS), a System/36, a System/38, or a 5520. It uses advanced program-to-program communications (APPC) or advanced peer-to-peer networking (APPN), depending on the options specified when creating the communications objects. (The article *Advanced Peer-to-Peer Networking* details the advantages of using APPN.) The system also contains support to communicate with a remote spooling communications subsystem (RSCS) for mail sent to System/370 Professional Office System (VM/PROFS) users.

Personal Computer Integration

One of the most important elements to a successfully integrated office is integrating the personal computers rapidly populating the office.

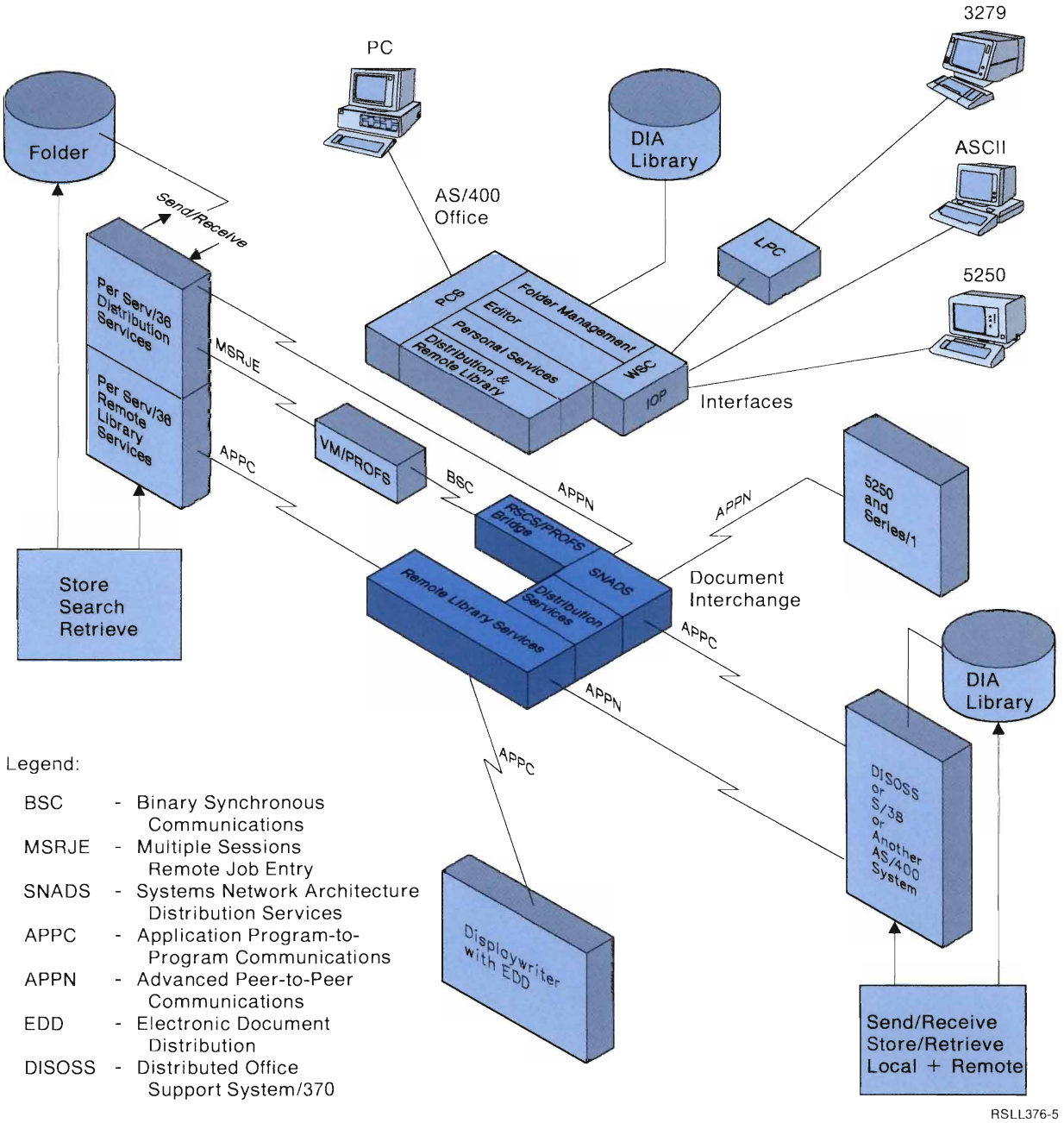


Figure 3 Electronic Mailing System

The AS/400 system improves function, performance, security, data integrity, and data sharing, and expands the application base available to the PC user, by integrating this support into OS/400.

Figure 4 shows the wide range of services that make connection of a personal computer to the AS/400 system very appealing. These services are provided by functions, called host servers, that communicate to the attached personal computer through a work station controller or communications attachments within the host system.

Host Processing

The most important services provided by the host servers include shared folders, virtual print, file transfer, and distributed data management (DDM).

Shared folders is the critical link that provides the data-sharing benefits described. This link allows the attached personal computer to share its objects with the AS/400 system. The value of system facilities, such as single-level storage, integrated data base, security, and communications, can be added to the capabilities of the personal computer. The host server maps PC functions to equivalent host system functions transparent to the user. For example, using the DIR command on an attached personal computer issues a list of files, their size, and the date and time they were last modified. However, this list can be large, making it difficult to find the desired file. Subdirectories can help to organize files, but may not break down the list sufficiently for some needs. When stored in an AS/400 shared folder, information about the file, such as author, description, or keywords, can be added without affecting the application. Using the search function of the integrated data base, AS/400 Office can provide a list of documents written by a certain author, or files with a specific description.

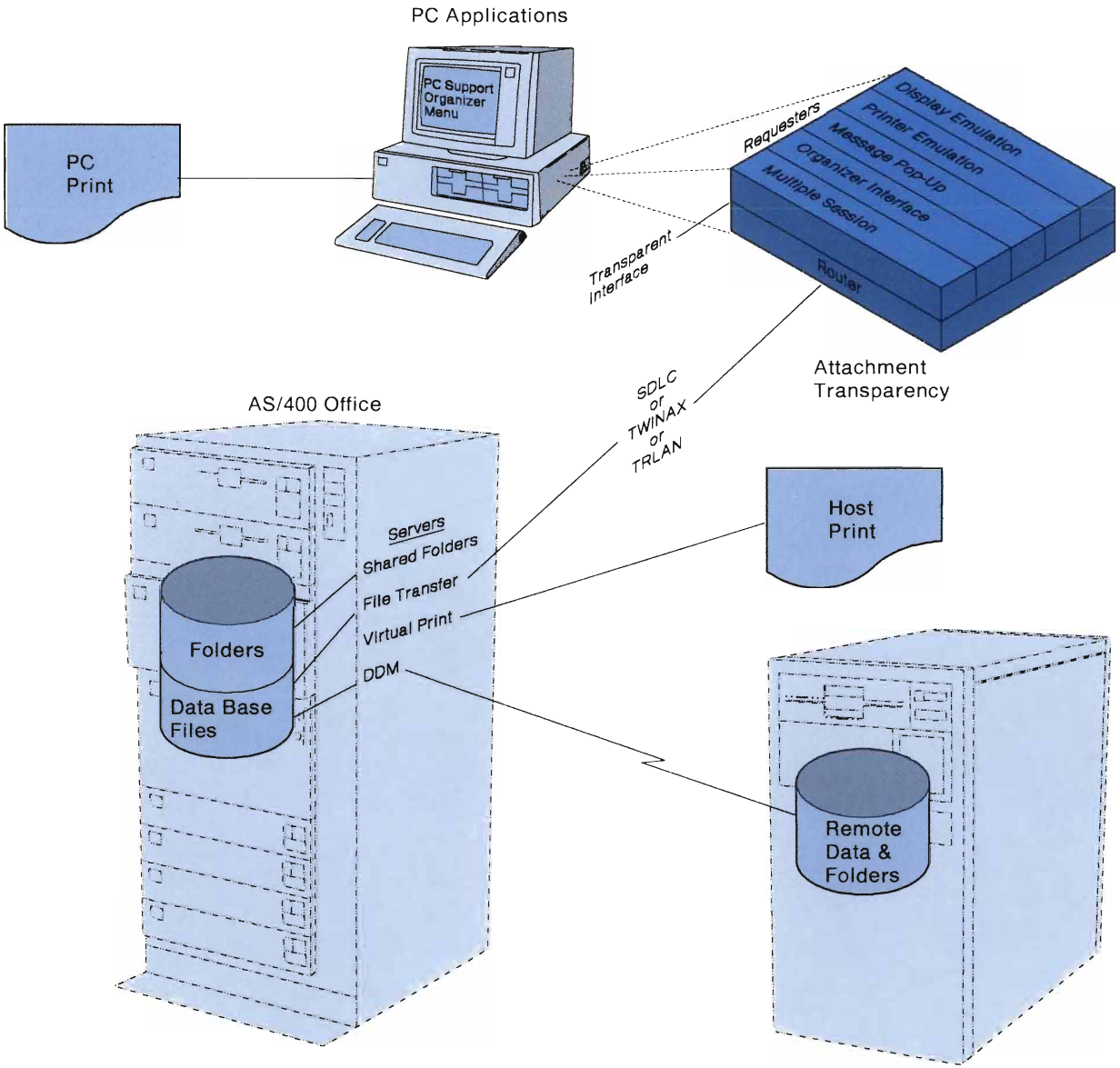


Figure 4 PC Integration (Requester/Server)

The list can be derived from a folder or from a full DIA library search.

Another service, virtual print, allows a PC application to use a host system printer as if it were attached to the personal computer. File transfer provides the capability to transfer files to and from the AS/400 system. In the process of transferring the files, they are converted to ASCII, BASIC Sequential, BASIC Random, DIF, DOS Random, or EBCDIC. This allows almost any PC application to retrieve data from the AS/400 system.

Finally, DDM provides an interface that allows a PC application to retrieve data from the AS/400 system it is connected to or from any other AS/400 system in the network. DDM makes the location completely transparent to the application. Using APPN, the system selects the best route and handles the data transfer.

Attached PC Processing

The attached PC requester communicates through a router that provides attachment independence to all applications running on the personal computer. The router converts the communications request to the correct connectivity option for the attached personal computer. The services provided by attached PC processing include: display and printer emulation, the PC Support Organizer menu, message pop-up, and multiple sessions. The attached PC requesters are all designed with user-friendly PC interface techniques available (pop-up help windows, action bars, and the like).

Display emulation is the primary function allowing the attached personal computer to have access to all of the AS/400 system function. It does this by making the attached personal computer appear to the system as a host-dependent work station. The printer emulation function allows host system printing at the personal computer much like a host-dependent printer.

The PC Support Organizer menu shows both AS/400 system and PC applications. It comes with a base set of office applications and is extendable to allow users to add other frequently used applications. With the PC Support Organizer menu, it is no longer necessary to use the hot-key sequence between systems to run applications. The user can select the application from the menu and is not required to know if it is running on the attached personal computer or the AS/400 system. This is achieved using the shared folder support of the filing system. By designating a shared folder as a PC disk drive (a virtual drive) and copying the applications to this drive, the applications have access to all of the shared folder services. With this feature, the AS/400 system can act as a file server for PC users.

Message pop-up displays messages sent from the AS/400 system in a pop-up window on the attached personal computer. This allows the user to take appropriate action without interrupting application processing.

And finally, multiple sessions provides up to five active sessions on the attached personal computer. This allows communications with different host systems, or multiple sign-ons to the same host system, all using the same connection, with all sessions active at the same time. In addition to using this capability for normal business activity, it can be used for central network management. A system operator can control a network by signing on remote systems; APPN performs intermediate routing.

The attached PC user running a PC application views the AS/400 Office filing system as a hard disk (with many restrictions removed). The system is capable of storing gigabytes of data for the user. Users can share this data with anyone they authorize throughout the entire network. Additionally, file locking is provided at the byte level to allow PC applications to share these files and lock only the record, or part of the record, being updated. As stated, the filing system supports the PC naming conventions. This means that commands like PC Directory (DIR), Making Directories (MD), Changing Directories (CD), and all of the functions within subdirectories, work transparently.

Flexible and Powerful Editors

The AS/400 system features the AS/400 Office editor and DisplayWrite 4, and provides an editor-of-choice option. When a personal computer is attached to the AS/400 system, either editor may be used. The AS/400 Office editor is compatible with the DisplayWrite 4 editor, and they can exchange documents very well. A user can take advantage of the improved processing techniques available with the AS/400 Office editor while working with DisplayWrite 4 documents. The inverse is also true. AS/400 Office editor documents can be shared with a DisplayWrite 4 user by placing them in a folder. The document structures and many of the typing techniques are identical between these two editors. Figure 5

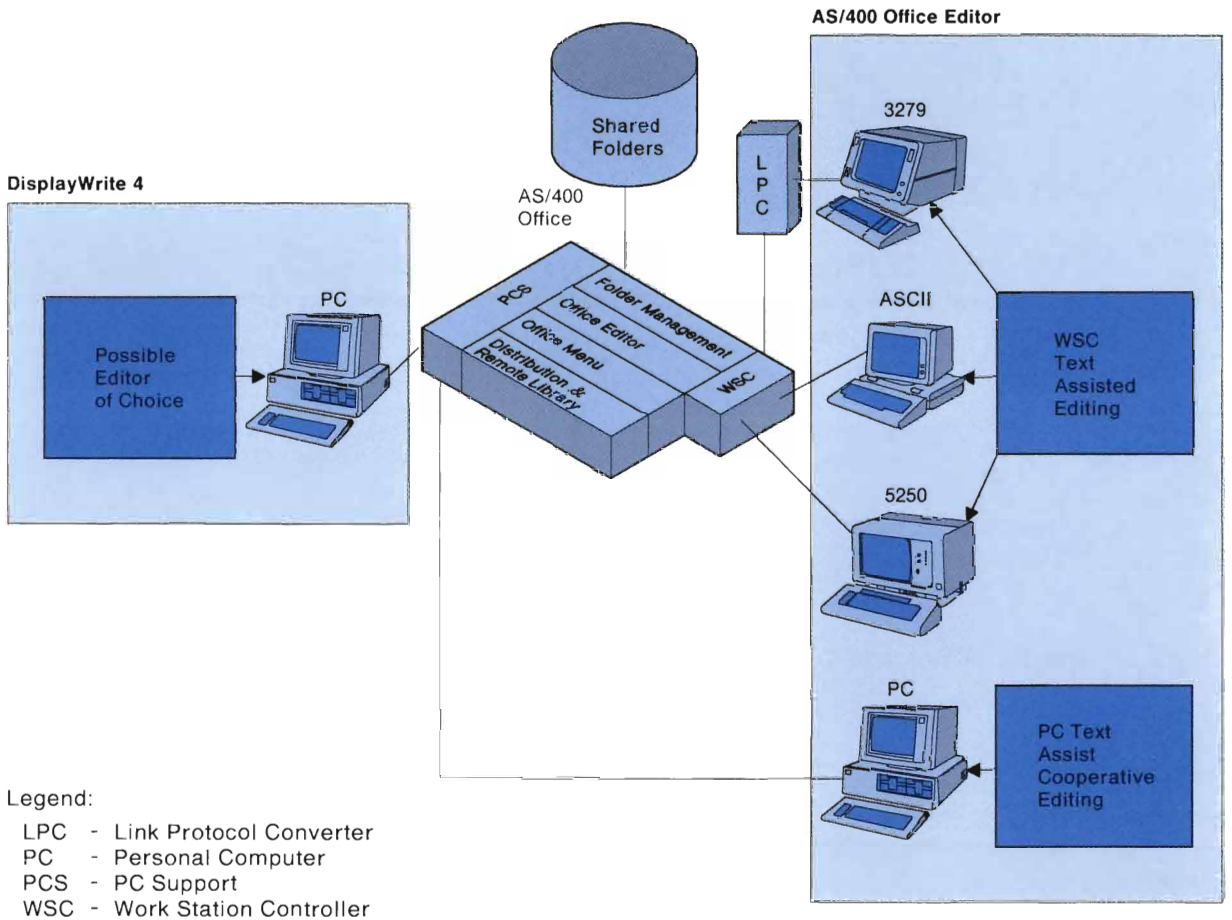


Figure 5 Editor Attachments

shows how the editors relate to the AS/400 work station attachments.

Although appearing very similar, several differences should be noted between an editor that runs on a personal computer while unattached and one that has the advantage of the powerful AS/400 System Processor while attached. The AS/400 Office editor supports the entire range of work stations that can be attached to the AS/400 system, and as work station hardware improves, the editor improves. Some examples of improvements through the work station family include: block cursor on the scale line above the actual cursor; fast cursor-move for word, line, and paragraph; wide display size (27 lines, 132 columns); and color displays.

The editor's performance is very dependent on the processor and peripherals performing the actual work. If the editor is host-system based, expected response time would be similar to other host applications; if the editor is PC based, response time depends on the power of the PC processor. The AS/400 Office editor was designed to take advantage of cooperative processing technology using two innovative approaches. The result is an average response time that is much better than the average system response time for host-dependent work stations, and better than typical PC response times when doing processor-intensive activities.

The first approach, called work station controller text assist, relieves the host system of keystroke processing tasks associated with host-dependent work stations. The work station controller runs in a separate input/output (I/O) processor. The work station controller and the editor, through a dialog, allow significant processing to take place on the I/O processor, off-loading it from the System Processor. Figure 6 shows the editing function

Action	Work Station Controller Text Assist Functions	AS/400 Functions
Status/Scale Lines	Updates status line: audit window, pitch, and line number	
Tabs	Processes right, center, decimal, comma, and colon tabs during typing	Formats text for viewing on the display based on tabs
Word Spill	Spills words to next line when characters reach right margin	Adjusts line endings in changed text
Locate	Locates characters on display when "Find Character" mode active	Locates character when not found in current display
Horizontal/Vertical Scrolling		Shows next display based on cursor
Move/Copy	Makes first mark and prompts for end of block	Prompts for target and copies/moves marked text
Delete	Makes first mark, prompts for end of block, and deletes within lines	Deletes marked text not delayed by work station controller

RSLL319-4

Figure 6 Cooperative Processing: Work Station Controller Text Assist

split between the work station controller and the host system, using work station controller text assist.

The second approach is PC text assist. This uses PC storage to provide an increased buffer area for text. It improves the average response time experienced by the user because it allows page-based functions to be handled on the attached personal computer, using the personal computer's own processing power, while implementing document-based functions on the host system. PC processing includes moving, copying, and deleting text, as well as adjusting line endings. Figure 7 shows the division of editing function between the attached personal computer and the host system.

For both techniques, the AS/400 system stores the documents and remains the primary processor for many functions that require the host system's resources and processing capability. Compute-intensive activities, like pagination, printing, spelling verification, and query, are processed on the host system. The host system also handles work that would otherwise require large amounts of data transfer, like data merge for editing or printing. This cooperation allows the processing to be done more efficiently (shown in Figure 8).

Action	PC Text Assist Functions	AS/400 Functions
Status/Scale Lines	Updates status line: audit window, typestyle and pitch, context and line number	
Tabs	Formats text for viewing on the display and processes right, center, decimal, comma, and colon tabs during typing	
Word Spill	Spills words to next line when typing and cursor reaches right margin, and adjusts line endings in changed text	
Locate	Locates characters on page when "Find Character" mode active	Locates characters not found in current page buffer
Horizontal/Vertical Scrolling	Refreshes current page buffer when the cursor moves out of the current page	Refreshes the display from page buffer based on cursor movement
Move/Copy	Makes both first and last mark, prompts for end of block and target, and copies/moves the text if within a page	Moves/copies/deletes marked text that is not in the current page buffer
Delete	Makes both first and last mark, prompts for end of block, and deletes the text if within a single page	Deletes marked text that is not in the current page buffer

RSLL320-4

Figure 7 Cooperative Processing. PC Text Assist

Additional Aspects of Integration

In addition to the integrated use of system functions that provides the basic underpinnings of support, most of which are hidden from the user, the AS/400 Office product integrates function from two more visible standpoints. The first is the diverse functions available while using the editor, and the second is document access and interchange through folder management services. These are highlighted in Figure 9.

The office word processing function is the base for all text operations. Therefore, the AS/400 Office editor becomes a very important element of integration, due to the internal interfaces provided for various office functions. If the Office mail function needs to edit, view, or print a note, the internal interface provided by the editor is used to accomplish the task. Office word processing also often requires users to merge data into a document while editing or printing. The interface to AS/400 Query allows merging while editing and, using the RUN instruction within the office edit function, inserting the output of an application program running on the AS/400 system directly into a document at print time.

Folder management services is the basic element of the document access and interchange functions that provide application independence from the stored data type (see Figure 9). This independence is achieved by detecting discrepancies between the stored document type and the requested document type when the document is accessed. Document conversions are automatically performed before any further requests are processed. This ensures the document is in the correct format for processing, while freeing each office service from the burden of managing multiple data types. The result is a consistent DIA library and folder interface.

The Office editor uses a high-level document access method. This provides the Office editor with a data management facility that works specifically with documents. This data management allows partial-document access, minimizing unnecessary data movement by allowing the editor to get, put, or replace a portion of a document. A single page can be updated without first making a complete temporary work copy of the entire document. The editor can go to specific lines and pages of a document, find

Function	Entirely Host-Based	Entirely PC-Based	Cooperative Environment	
			Host	PC
Performance	Average	Average	Good	Good
User Interface	Average	Good		Good
Paginate, Spell, Print	Good	Average	Good	
Sharing of Data	Good	Poor	Good	Good
Integration	Good	Average	Good	

RSLL321-3

Figure 8 The Cooperative Processing Advantage: Doing the Work Where It Is Most Efficiently Done

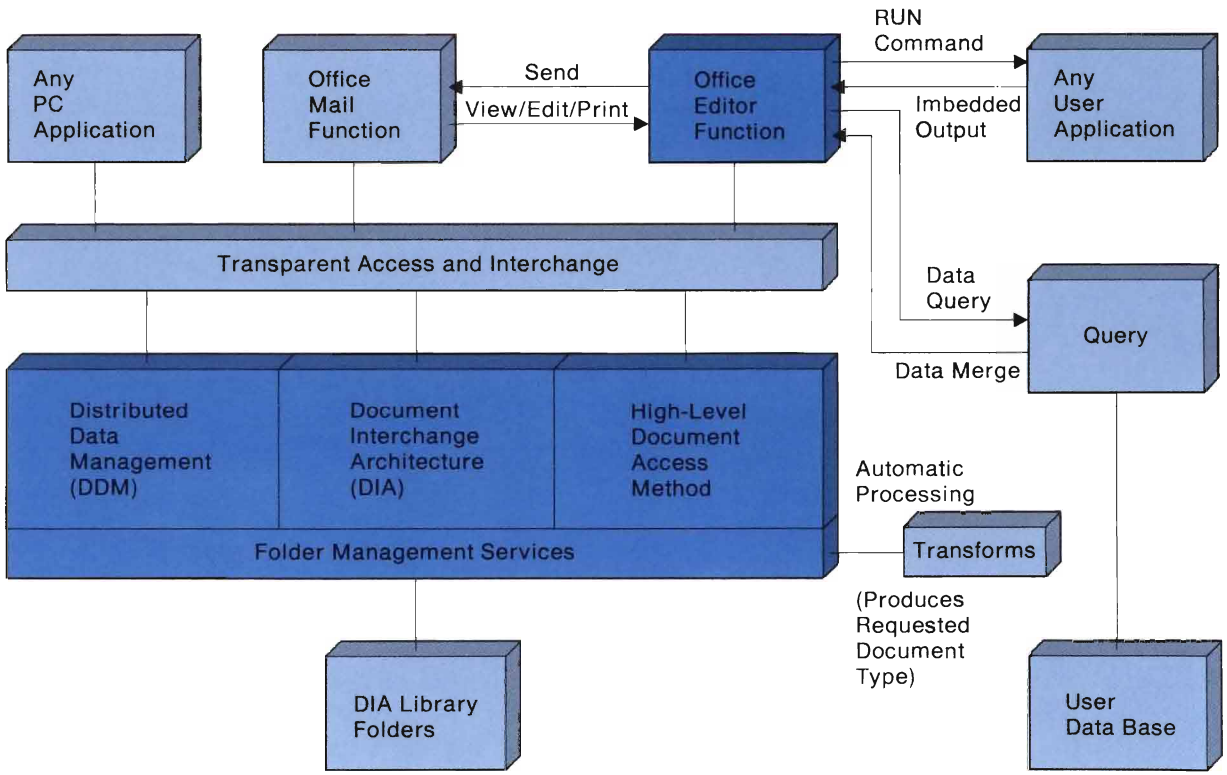
currently active page formats, and move, insert and delete text, while removed from the specifics of how the document is stored. This access is supported for both host-system and PC editors. The host-system editor requires access by line and page reference, while the PC editor requires access by byte offset and length.

The DIA library and folder interfaces are also application program interfaces (API) that follow the strategic Systems Application Architecture™ (SAA™) standards. These interfaces are available to users wishing direct access to AS/400 functions. Because these interfaces follow SAA standards, applications written using them can be used on other systems supporting SAA applications. An example is the Structured Query Language (SQL) interface to the file transfer function. It allows a PC application to write SQL statements to get data from any AS/400 data base in the network.

AS/400 Office Menu

The AS/400 Office menu is the user interface to Office functions. Figure 10 shows the menu interface, designed with office activity in mind. The information shown is comprehensive, including options, a command line, informational messages, and the monthly calendar. The menu options provide access to the diverse functions supported by Office, including: sending notes, messages, or mail (or anything contained in a folder); managing calendars with many scheduling options; organizing directories and distribution lists; and many more office functions. Also shown is an option that allows the user to customize the interface using additional menus. APIs have been strategically located to allow program access to certain Office functions. Separate functions can be combined or accessed in a different way to accommodate the office user.

Also noteworthy is the multiple suspend-and-resume capability. In a busy office, the ability to



RSLL417-3

Figure 9 System and Application Integration

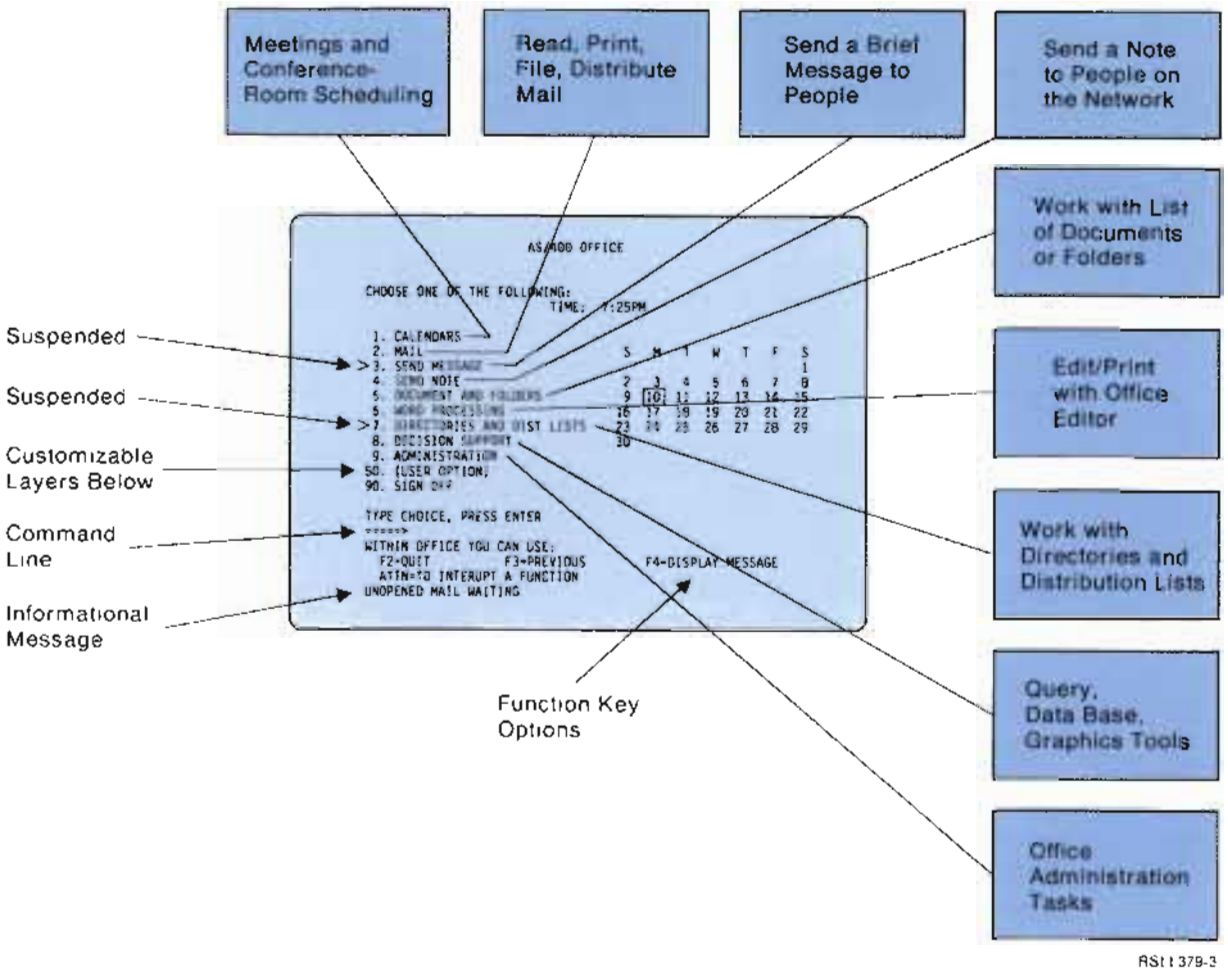


Figure 10 AS/400 Office Menu

interrupt an activity to do something else, and then return to the original task, is a necessity. This function suspends an activity using a single key, indicates suspended activities on the menu, and allows the activities to be resumed in any order.

Conclusions

AS/400 Office creates a truly integrated office environment for the IBM customer. It allows office work to flow smoothly and efficiently in an office that may include several different IBM systems. Users may perform the work on the work station with which they are most comfortable, while having available the AS/400 performance, usability, functionality, and integration necessary for a productive office environment.

By following the strategic SAA architectures, AS/400 Office is positioned for future participation as more systems adopt and support the architectures.

Acknowledgements

The authors would like to acknowledge the contributions of the following people to this article:

John C. Endicott, Karl C. Hanson, Timothy L. Kramer, Myron L. Anderson, and C. David Truxal.

References

1. **Document Interchange Architecture: Technical Reference**, SC23-0781-0, First Edition. May, 1985.
2. **Document Interchange Architecture: Interchange Document Profile Reference**, SC23-0764-1, Second Edition. May, 1985.
3. **Document Content Architecture: Final-Form-Text Reference**, SC23-0757-1, Second Edition. May, 1985.
4. **Document Content Architecture: Revisable-Form-Text Reference**, SC23-0758-1, First Edition. June, 1983.

Security

Discusses integrating security capabilities into the machine interface and the operating system to form a security model with the flexibility necessary in a changing user environment.

Wayne O. Evans and Richard J. Lindner

Introduction

As time progresses, a users' need for system security function changes. Initially, the user's need is to establish information asset protection practices with the AS/400™ system similar to those they are currently using. In the future, as applications become more sophisticated and the user requires data protection controls not apparent today, the system security features must be flexible enough to provide solutions for the protection problems encountered.

A system-wide AS/400 security model integrates the best security features of the System/36 and System/38. An object-oriented security implementation was selected as a base for the AS/400 security model. The machine interface uniformly enforces security for all references to objects, including libraries, files, and data. This increases the effectiveness of the data protection and cannot be circumvented. Like other security systems, AS/400 security is designed to protect the data from unauthorized disclosure or modification. Passwords are used to validate user access to the system. The individual user attributes are stored in user profiles.

The implementation challenge of AS/400 security was to provide a wide range of security functions that met the following goals:

- It had to be able to run without security, with limited security, or with full security as configured by the user's installation, while the underlying security functions built into the machine interface were always active.
- It had to allow resources that do not yet exist to have security definitions in an architecture that is dependent on object existence prior to authorization.
- It had to allow individual authorities to override the public authorities and control access to data.
- It had to allow grouping of authority based on relationships (for example, groups of users and lists of objects) adapted to the structure of the organization using the system.

The AS/400 security model required solutions within Operating System/400™ (OS/400)™, as well as in the underlying machine interface. The AS/400 implementation of multiple levels of system security, user profile attributes, methods of grouping authority, authority holders, program adoption of authority, and the authority search order facilitates solutions to the various needs of today and the future.

System Security Levels

The AS/400 system is designed for businesses where the security requirements range from no security to full security. The system can be configured for three levels of increased security: physical security only, sign-on security, and full resource security. When physical security is adequate, the AS/400 user does not need to be enrolled, meaning that when a user has physical access to the system, the user is allowed to sign on and use the system. Security does not restrict access to any system objects; however, users are prevented from affecting the jobs of other users in the system. When sign-on security is required, the AS/400 user must be enrolled and must enter a valid password for access. If access is granted, users are not restricted from any function, similar to a physical security system, unless the user profile indicates the user is a limited-capability user. A limited-capability user is limited to selecting options from menus, and to using a limited set of commands that allows messages to be sent and viewed, the status of the user's job to be displayed, and the ability to sign off the system. When full-resource security is needed, the AS/400 user must be enrolled and enter the correct password. Resource security can restrict data access to authorized users. Individual users can be authorized to use system objects such as files, programs, devices, and commands. A user can also be enrolled as a limited-capability user when full resource security is employed.

The machine interface validates security when an object is accessed, so levels of security were implemented totally within OS/400. Advances added to allow users to select the level of security for the system are:

- Allow system access to users that are not enrolled.

  When the system is configured for physical security, the AS/400 Sign-On display asks for a user name but no password. The user name is used to search for a matching user profile. If no profile is found, the operating system creates a user profile for the user. This dynamic creation

of a user profile satisfies the machine interface requirement that every job have a user profile. The created user profile is not deleted when the user signs off, anticipating the same user will use the system again. This design also allows users to customize their profile and easily move to a more secure level.

- Allow unrestricted access to objects.

  Unrestricted access to objects is implemented using a special all-object authority (*ALLOBJ) that modifies the processing of machine instructions to eliminate any authority checking when accessing objects. OS/400 uses all-object authority as the default value for user profiles in a system configured with physical or sign-on security.

When changing to full-resource security, the system removes the *ALLOBJ authority from all user profiles except users in the security officer class. Removing *ALLOBJ special authority indicates the user is to be controlled by authority to objects and causes the machine interface to check the user's authority to access objects.

User Profile Attributes
The user profile is the object that identifies the user to the system. The primary use of the user profile is to store security-related information.

User Class
The user class defines the operations a user can perform based on the task assigned to the user. AS/400 security has five hierarchical user classes. A user class can perform all of the functions for lower user classes. The user classes and functions, ranging in order from increased level of access to lower level of access, are:

- Security Officer, who performs all security functions including creating security administrators.
- Security Administrator, who enrolls users and secures resources. Security administrators can be assigned for functional areas; the security administrator for one area cannot remove or enroll users in other functional areas.
- System Programmer, who performs application development functions. Application programmers can be restricted from modifying production applications.
- System Operator, who performs all system operation options. So that a system operator can back up the system, operators are allowed to save and restore objects they are not authorized to use.
- Work Station User, who performs application functions.

In addition to controlling the functions allowed, the user class controls the menu options that are available to the user. Easy-to-use menu interfaces are provided for system functions, so that particular menu options are presented to a user based on the assigned user class. The AS/400 system has a simplified enrollment process that uses the user class to determine the special authorities granted to a user.

The security officer and security administrator user classes are granted access to privileged machine instructions to operate on user profiles. This means that administrators can enroll users when they have the security administrator (*SECADM) special authority in their AS/400 user profile. The machine interface security validates the user's authorization to the commands and privileged instructions, such as creation of user profiles, system backup and restore, and operations on other user jobs.

Limited-Capability User
Most system menus provide a command entry line. To restrict use of the command entry line and limit the user to selecting menu options, the AS/400 user can be enrolled as a limited-capability user. The IBM and user-defined commands allowed by a limited-capability user can be individually defined. The IBM commands initially available to the limited-capability user are End Session (SIGNOFF), Display Job (DSPJOB), Send Message (SNDMSG), and Display Message (DSPMSG). Limited capability also determines the fields that a user can modify at sign on.

Objects Owned and Authorized
The user profile records all the objects owned and objects authorized to the user. The user that creates an object becomes the owner of the object. The list of all objects owned by a user is stored in the user profile. A second list is all the objects that have been authorized to the user profile. The virtual address of the objects is stored rather than the object name. The virtual address optimizes the machine-interface authority-checking algorithm because the machine instructions use virtual pointers when referring to the objects.

Authorization of Objects
The owning user profile has the authority to grant other users authority, and the public authority to use an object.

AS/400 security is designed with individual object authorization; eight authorities can be granted to a user. These authorities are independent (not hierarchical), however, for some operations a combination of authorities is required.

- Object management (*OBJMGT): Rename, move, or authorize other users.
- Object existence (*OBJEXIST): Delete object.
- Object operational (*OBJOPER): View description of object.

- Authorization List Management (*AUTLMGT): Authorize users on an authorization list.
- *READ: Read.
- *ADD: Insert new entries (for example, records, messages, and objects).
- Update (*UPD): Modify existing entries.
- Delete (*DLT): Remove individual entries (records, messages).

These authorities can be granted or revoked individually, though, to simplify security administration, four additional terms represent multiple individual authorities.

- *ALL: Full authority for object, including capability to manage security on objects except authorization lists.
- *CHANGE: Authorities to use and modify, but not delete.
- *USE: Authority to use, but not modify.
- *EXCLUDE: No authority. The user is restricted from any use of the object.

The use of independent authorities was implemented because this provides precise control over the functions an individual user can perform. Descriptive terms such as *CHANGE and *USE are used to improve the understanding of a user's capability.

Methods of Grouping Authority

Grouping techniques eliminate the need to repeatedly specify authority when several objects or users have the same authority. This also reduces the number of authorities that must be stored by the system. Authorization lists and group profiles are implemented to simplify security administration. Performing a single operation can grant a user authority to multiple objects, or multiple users authority to a single object.

AS/400 Authorization Lists

An authorization list allows the authorization of multiple objects to be controlled by a list of users, each with a specific level of authority for all objects referring to the list. When an object is secured using an authorization list, the security kernel searches for specific authority to the object and then for authority on the authorization list.

Figure 1 shows an authorization list that secures several objects. When a user (USER1, USER2, or USER3) is on the authorization list, the user has that authority to all objects secured by the authorization list. When a new object is secured by the authorization list, all users on the authorization list have authority to the object, thus eliminating the need to specify the authorities individually. If a user has private authority to an object, the private authority overrides the authority on the authorization list. For example, USER2 has the private authority *USE to file B, which limits access to read operations on this file, even though USER2 has *ALL authority to the other objects controlled by the list.

Controlling authority of users on the authorization list requires a new authority concept, the concept of authorization list management (*AUTLMGT). Object management (*OBJMGT) authority indicates the user is allowed to authorize the objects to others but does not control the authority of users

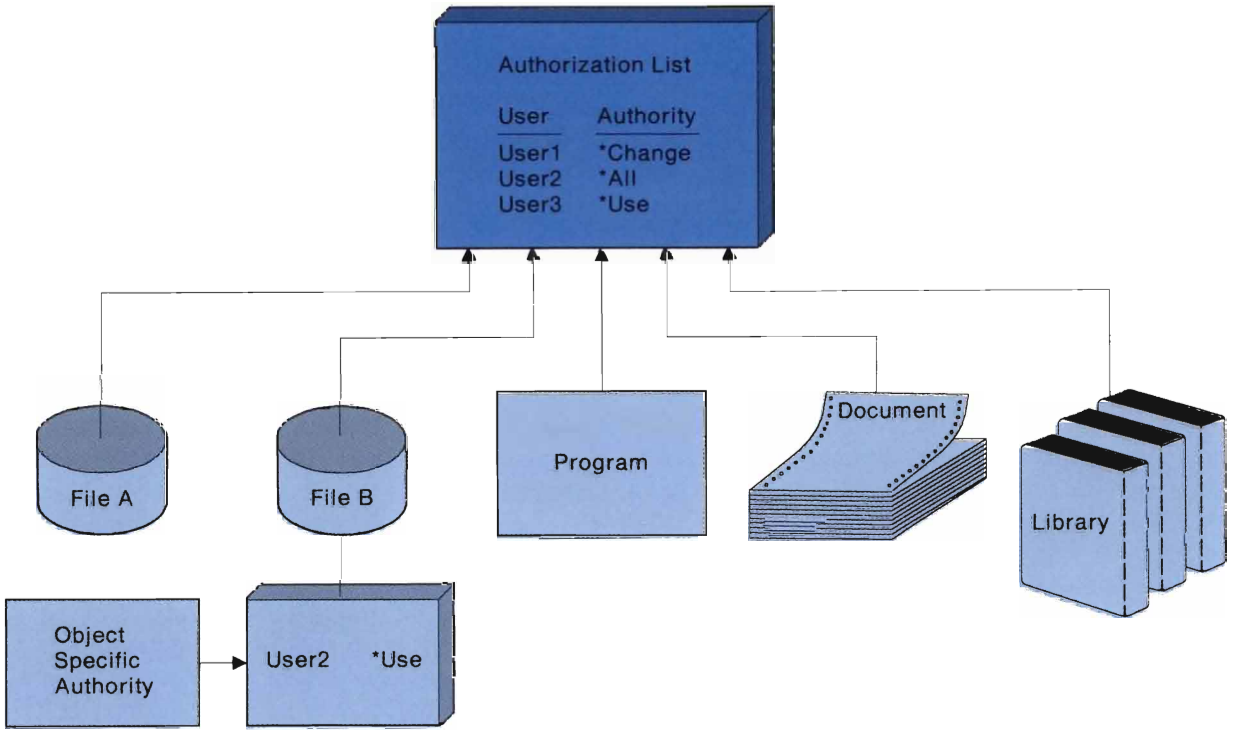


Figure 1 Authority List Securing Multiple Objects

on the authorization list. Authorization list management (*AUTLMGT) is required to change the list authority of users to the objects controlled by the list. The owner of the authorization list and users with *AUTLMGT authority can add, remove, and change users and their authority on the authorization list.

The AS/400 machine interface implements the authorization list as an object type. The machine interface provides the underlying support for this concept by implementing the authorization list management interface that allows modification of user authorities that appear on an authorization list.

AS/400 Group Profiles

Group profiles allow users to share the authority of another profile and have their own user profile and passwords. This allows members of the same department to share common objects (such as programs and data). The authorization for all members of the group can be managed by authorizing the group profile. The group profile simplifies adding new profiles or removing profiles as different users join or leave a group.

A user that is a member of a group has the option of having OS/400 automatically grant the group profile authority or transfer ownership to the group profile for objects created by that user. Transferring ownership or granting authority to the group profile authorizes all group members to the object.

When an individual user profile is authorized to an object, that private authorization overrides the authority of the group profile. Individual user authority allows specific group members to be given either less or more authority to an object than other members of the group.

Figure 2 illustrates three user profiles that have the same group profile. The user profiles in the group share the object authorities of the group profile, D46E. Individual profiles (JONES, EVANS) in the group have individual authorities that are not shared with the members of the group. For example, the profile EVANS has been granted *USE authority to the program B. This allows EVANS to run the program, but not delete the program, because it is owned by JONES. The group profile D46E and JONES both have authority to file D. But, the private authority of *USE for user JONES limits access to *USE (read) operations even though other group members have *CHANGE authority. This is an example of how private authority overrides the authority of the group.

Authority Holders

When a user profile is authorized to a file, the AS/400 machine interface stores a pointer to its object and the authority with the user profile. Deleting a file causes all record of the authority to the file to be removed from the system by the machine interface. To support authorization for non-existent files, which is used in applications migrated from the System/36, AS/400 security implemented a file called an authority holder. The authority holder is a shadow object that holds the authorities of a file when the object does not exist. An authority holder can be created for a file and users can be authorized before the file or library exists. When a file is created with a name that matches the authority holder, OS/400 attaches the authority holder to the file.

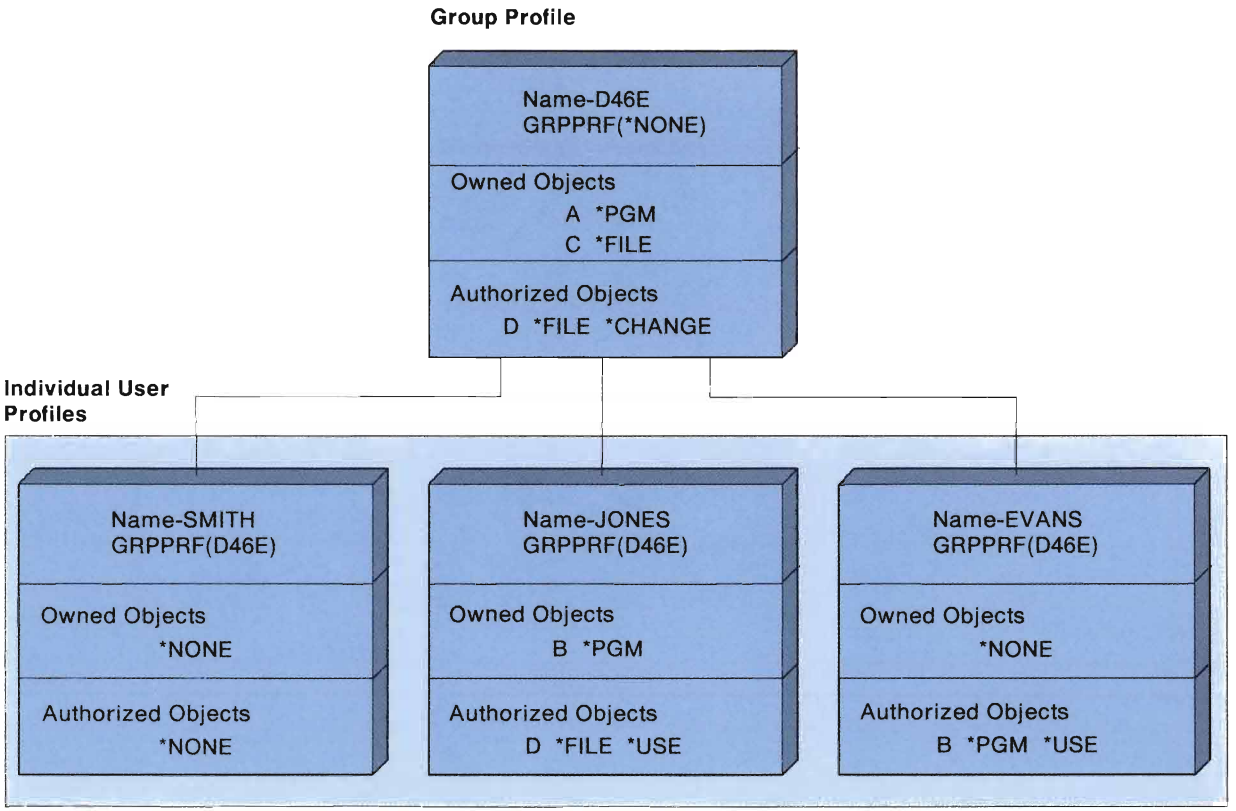


Figure 2 User and Group Profile Relationship

Figure 3 demonstrates the use of authority holders. No authority holder exists for file A, so when it is deleted the authorities are removed. File B has an authority holder; file B can be deleted and the users and their authority are retained. Granting a user authority to file B causes the user profile for that user to be authorized to the authority holder, rather than to the file object. Authority can be granted to file C even though the file does not yet exist because there is an authority holder for C. When file C is created, the system attaches the authority holder to the file. The authority holder name is simplified in Figure 3. The authority holder name includes both the file name and library name because the same file name can exist in multiple libraries.

Because authority for non-existing objects is only a requirement for migration, authority holders only apply to program-described files, rather than every type of file or other objects. Also, the command to create authority holders requires special authorization, thus giving the user control over the use of authority holders.

Program Adoption of Authority

Adoption of authority is used to allow a program to perform operations that require authority the user does not have. Rather than granting the user additional authority, the user application program calls a program that adopts the authority of the application owner to perform the operation. A program that adopts authority is identified during program creation and uses the owner's authority while the program is running.

The adoption of authority is useful when creating ease-of-use interfaces for application users. The users do not need to be authorized to objects; they need only to be authorized to the program that performs the needed tasks. This prevents the user from accidental or intentional misuse of resources. Because a program that adopts authority allows the user of the program to assume the authority of the owner, the system protects programs that adopt authority by restricting their transfer of ownership and restore operations.

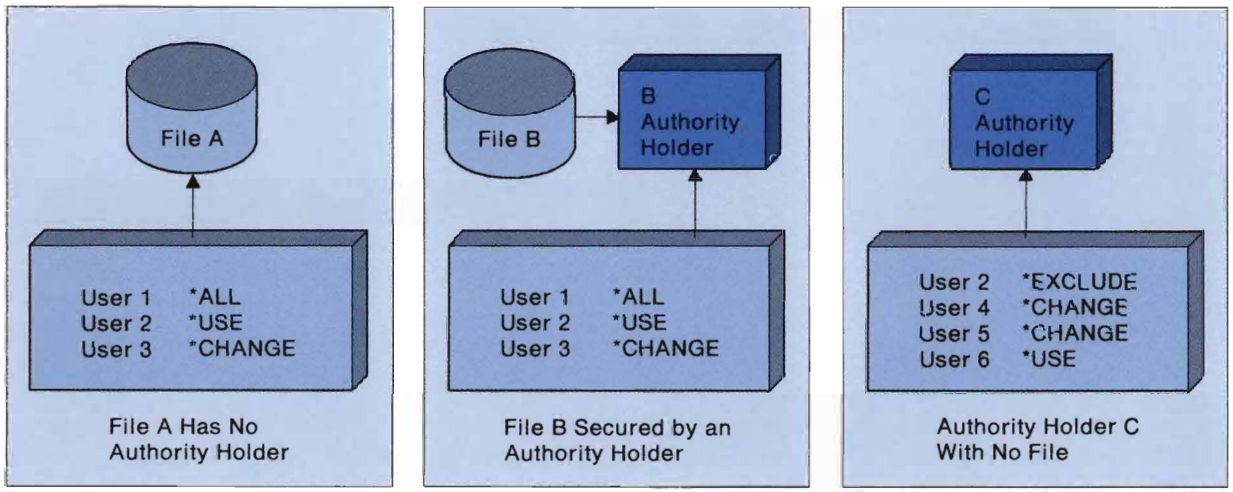


Figure 3 Authority Holder Usage

Adopted authority is passed to called programs, but the system prevents passing authority to programs that interrupt normal job processing. Authority is not passed during message break handling programs, attention key programs, or debugging breakpoints.

Authority Search

The AS/400 security model provides a specific authority search because it allows individuals to be given less authority than the public; that is, they may be excluded from the use of an object. When a user is authorized to an object, the user is limited to the authority assigned to the user. The default (public) authority only applies for users that were not authorized.

The search for authority is implemented in the machine interface to optimize performance (because the machine validates the user's authority). When the machine's security kernel checks a user's authority to access an object, program adopt authority is added to the first authority found when using this search order:

1. Individual User Profile.
   a. *ALLOBJ special authority allowing unrestricted access.
   b. Private authority to object.
   c. Private authority on the authorization list (if the object is secured by an authorization list).

2. Group Profile (if individual profile has a group profile).
   a. *ALLOBJ authority allowing unrestricted access.
   b. Private authority to object.
   c. Private authority on the authorization list (if the object is secured by an authorization list).

3. Public Authority.
   a. Public authority to object.
   b. Public authority on the authorization list (if the object is secured by an authorization list).

Searching the individual user profile first, and using the first authority found, allows users to be granted less (or more) authority for an object than the public, group profile, or authorization list authority. A new authority, *EXCLUDE, can be assigned to a user profile to restrict the user from access to an object. Adopted authority is additive to allow a user to perform operations on objects while running an adopted program that is normally restricted.

Conclusions

The AS/400 security model contains advanced features that provide solutions for the data protection problems of today and the future. Supporting the distinctly different underlying concepts of previous security models has enhanced the ability of users to migrate to the AS/400 system. The increased flexibility of this security model will satisfy security needs of users that migrate and future users with more complex security needs.

The implementation of this model by integrating security throughout the layers of the system capitalizes on the system strengths and optimizes the performance of its various functions. This implementation approach has also provided the constructs for expansion as the needs arise.

Electronic Customer Support

Describes integrated electronic customer support which brings an exciting new dimension to the partnership between the customer and IBM.

James R. Morcomb, Michael J. Snyder, Earl W. Emerick, and David L. Johnston

Introduction

The electronic customer support functions bring an exciting new dimension to the customer and IBM partnership. Many new easy-to-use system support functions have been brought to the fingertips of the user. These functions span a full spectrum of support, from hardware service and software problem analysis to marketing technical support and information.

To accomplish this on the AS/400 system, it was decided to integrate the support vehicles into a cohesive set of functions within Operating System/400™ (OS/400™). This set of electronic customer support functions includes both support components and end-user functions. The AS/400 objectives included decreasing the frequency of incidents on hardware and software that require support and providing more responsive support for those incidents that do occur. The design goals were to provide automated problem detection and analysis and to provide access to support functions at the time of failure. Integrating the electronic customer support function into the system achieves these objectives and design goals.

Electronic customer support provides a single point of contact for the user to obtain support, which results in improved response time and increased productivity for the user. Figure 1 shows that the system provides the user direct access to the electronic customer support functions through a display station attached to the system. The backbone of the system support functions is the built-in communications interface to the IBM Marketing and Service Support Systems. The integrated electronic customer support functions operate through a set of interfaces with the IBM systems applications to bring IBM support to every AS/400 display station.

Design Concepts

Two design concepts were deemed essential to enable the desired system management functions. The system components had to be designed so they are self-identifying, and they had to provide problem detection and analysis at the time of failure. Further, a new common input/output (I/O) architecture was needed to provide a consistent interface for new support-related functions.

Self Identification

The hardware and software components of the AS/400 system contain self-identifying information called vital product data. The hardware components contain within them their type, model, serial number, and load identifier. Type and model are used to identify hardware products and card features. The load identifier is the label of the initial program load (IPL) required to bring up the device or component. The serial number is a unique identifier for each component. Other component-specific information useful for using the component may also be provided. Software components contain their name and maintenance level. In addition, a header associated with each code component indicates loadable-code group membership, any dependencies on other hardware or software components, and supplier-identification information. The vital product data contained in the hardware and software is essential to support these functions:

- Error recording to the device or field-replaceable unit
- Automatic configuration and initial system build
- Feature ordering and verification
- Code-change management
- Service or configuration activity recording
- Code download prerequisite checking
- Failing unit location
- Service support entitlement

Collecting and storing vital product data is shown in Figure 2.

Problem Detection/Analysis

The system components are designed for problem detection and analysis at the time of failure. This emphasis on capturing data at the time of failure permits problem analysis for only a single occurrence of a failure. It avoids the use of failure re-creation techniques that are expensive to develop and often fail to detect and isolate intermittently occurring problems. Hardware configuration and information about how the

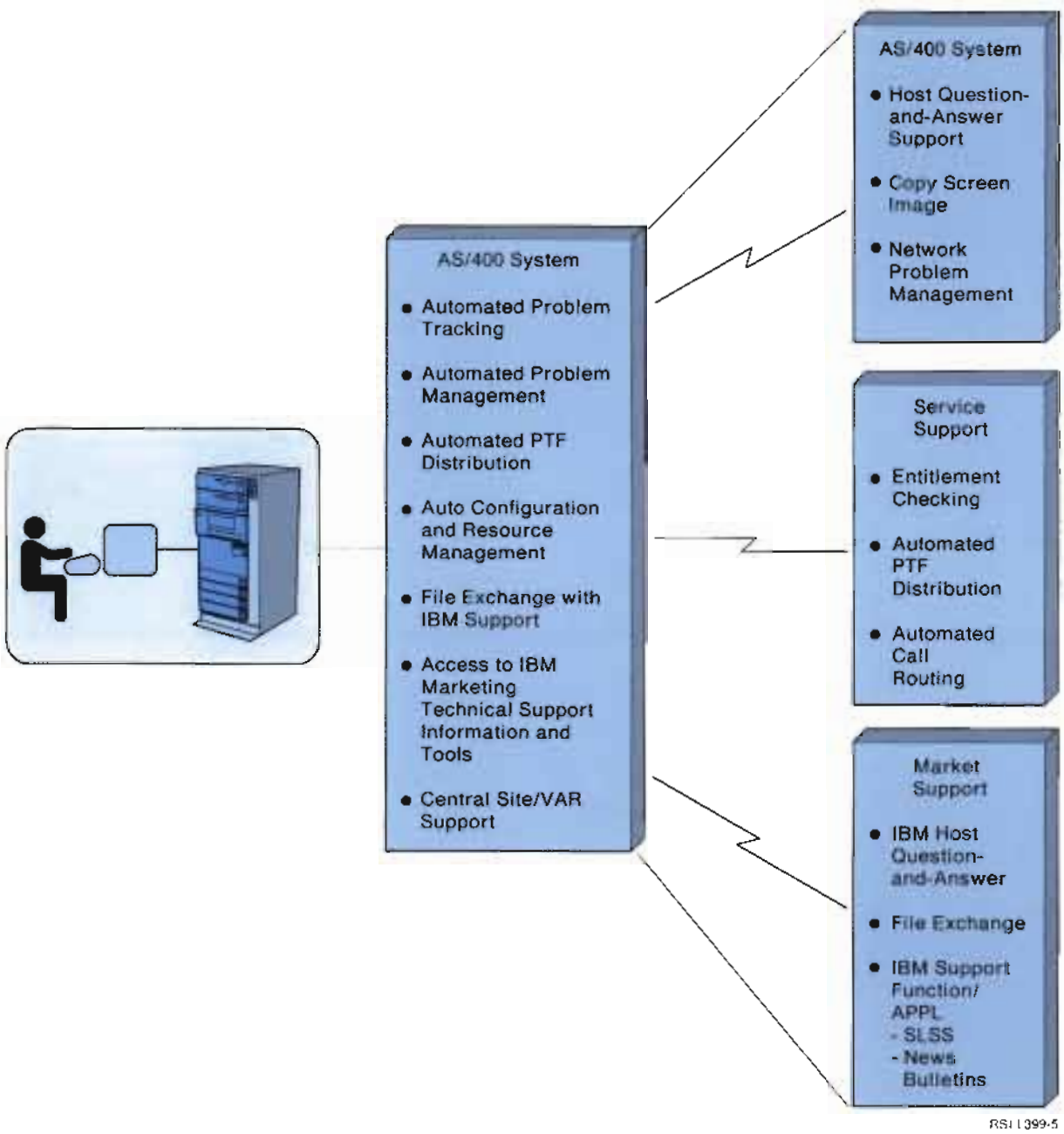


Figure 1 Overview of Electronic Customer Support Functions

hardware is being used is also captured so the system status at the time of the failure is known. This permits status-related problems to be more easily identified and diagnosed.

Common I/O Architecture

An internal interface, a common I/O architecture, has been designed for communications between bus-attached I/O devices and OS/400. It provides a consistent interface to all attached I/O processors for support-related functions. This approach has reduced the amount of operating system code required to provide the support functions and, more importantly, reduced the amount of component-specific code required in OS/400. This improves the ability to add new I/O processors to the system in the future.

Electronic customer support functions supported by the common I/O architecture are:

- First-failure data capture
- Vital product data collection
- Resource alteration notification
- Error notification
- Microcode download support
- Loading and running I/O processors or device tests
- Code debugging functions

Software components are provided within the I/O processors and within OS/400 to implement the I/O support interface between attached I/O devices and the operating system. The I/O processor components provide support-related functions

such as program debugging, code download, vital product data collection, configuration management, and data collection. The operating system components provide the error reporting path, an error log, a problem analysis and resolution driver, problem determination procedures, and configuration services. The common I/O architecture role in problem management functions is illustrated in Figure 3.

Support Components

The support components are: an error notification record built by the detector of the failure, a reference-code translate table that provides information about the actions that must be taken to resolve each failure, and software component descriptions that provide information needed for software management functions.

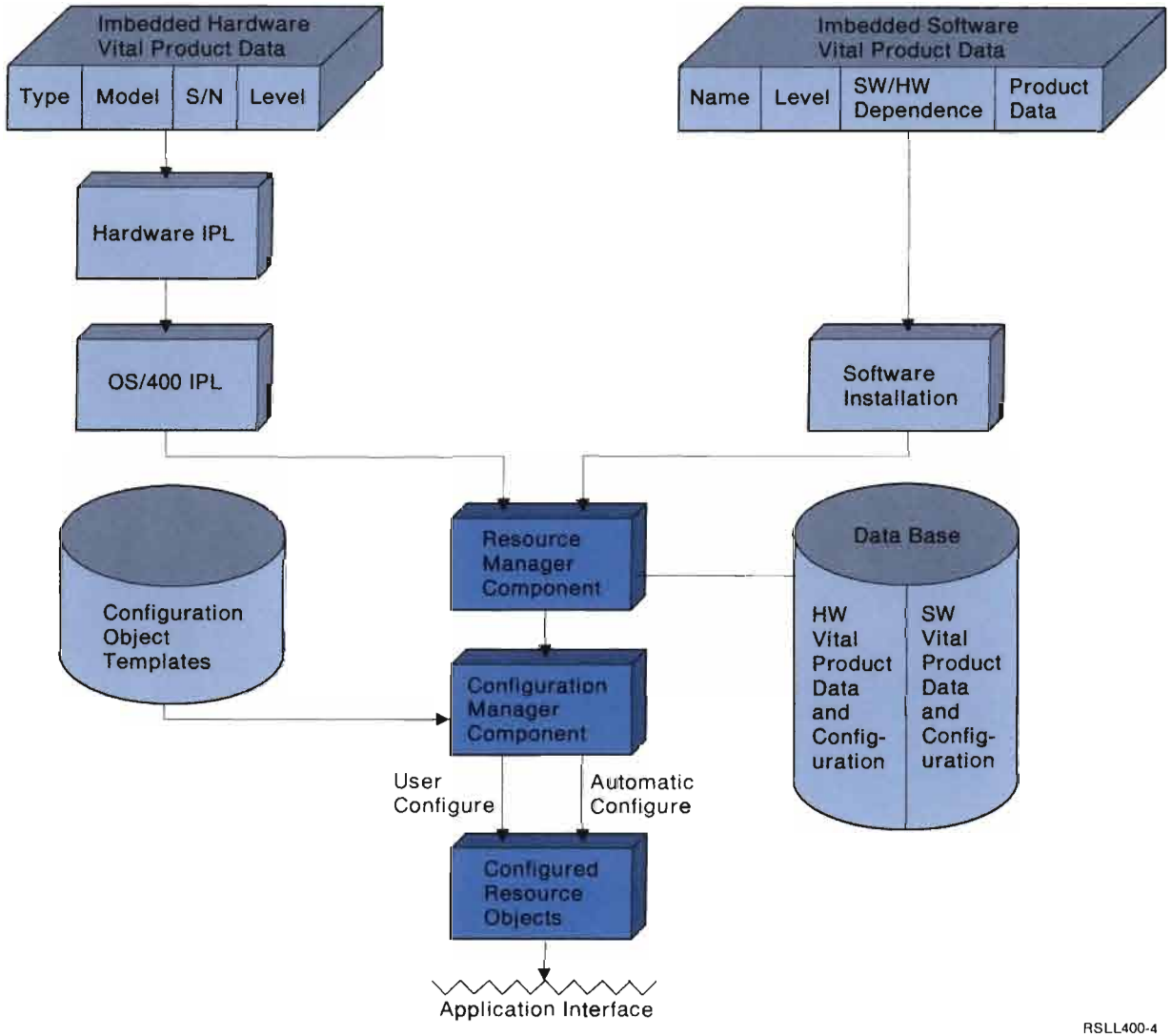


Figure 2 Resource and Configuration Management Functions

Error Notification

An error notification is built and logged for each detected failure. It contains the time-of-failure error information, and a pointer to hardware configuration information, which is added to the record before it is stored in the error log. It also contains the encoded name of the failure, called the reference code, and the name of the reference-code translate table used to evaluate the error condition. Thus, the error is encoded by the detector and the encoded name serves as an index to the reference-code translate table entry that contains information about the actions that must be taken by the system or the user to resolve the problem.

Reference-Code Translate Table

The reference-code translate table is a construct for enrolling system components in a structured set of problem analysis and resolution services provided by OS/400. The reference-code translate table is a data table that stores information about the set of errors the component can detect and the actions that must be taken by the system or the user to resolve each error. It contains no textual information, but contains pointers to the system message file that are used to access national language text describing detected conditions, user messages, and failing items. This provides an interface between electronic customer support and users in their native language, thereby enhancing user participation in system problem management. (Refer to *Software Design to Support National Languages* for more information on this subject.)

OS/400 uses this table to decode the error notification information into the set of problem analysis and resolution actions, and post-problem

resolution verification procedures that must be done by the system to resolve the problem. The reference-code translate table also contains alert code-point information, which is used by the alert manager component to build and send an alert. (An alert is a network problem management record sent to a problem management host system.) The code points are pointers to text phrases in a message file on the host system. In this way, many different systems in a network can communicate in a common problem-management language. The reference-code translate table role in problem-management functions is shown in Figure 3.

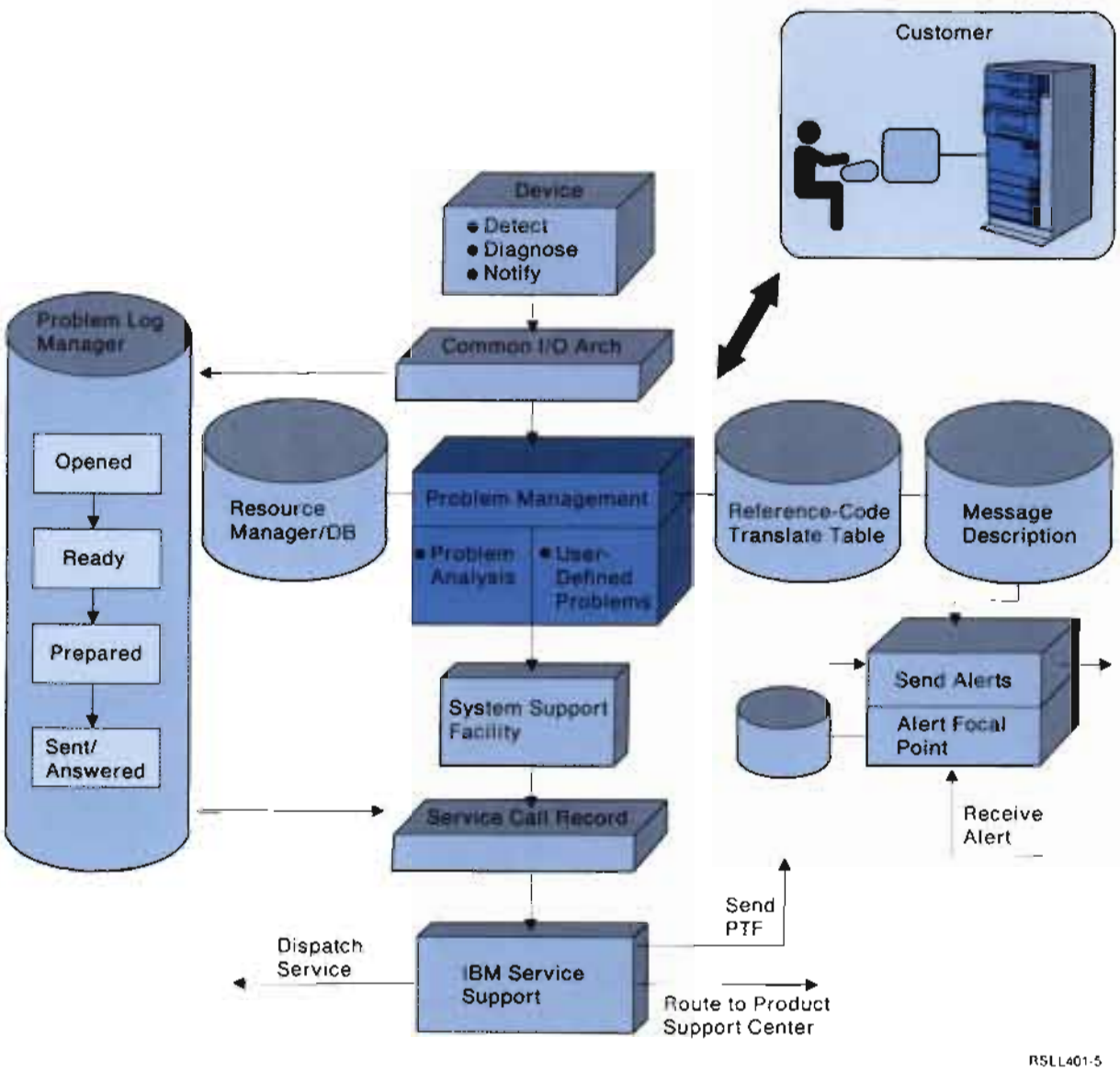


Figure 3 Problem Management Functions Overview

Software Component Descriptions

Software component descriptions define the product configuration and list the replaceable units of licensed program products and microcode groups. They identify all the major features and, optionally, loadable pieces of a software product and the code-replaceable units in each. The following information is also provided in these descriptions:

- The owner of the distribution and maintenance responsibility
- Hardware or software dependencies
- Replaceable unit list
- Maintenance level of replaceable units

These descriptions conform to a code-packaging architecture that permits a common set of system code management functions for all loadable code independent of code type. The software product description information is collected along with the software vital product data and stored in a system data base, as illustrated in Figure 2.

Resource and Configuration Management

The AS/400 system provides new functions for resource and configuration management. An overview of these functions is shown in Figure 2. The system resource manager component is the interface to the system resource data base. The resource data base is a read-only record of the system's hardware and software vital product data and configuration information. The information in this data base is made available

across the machine interface for use by application programs, problem determination procedures, and other operating system components.

The hardware vital product data is collected during each IPL or when resources are added after an IPL (when the device is powered on). The software installation manager collects and records software vital product data and configuration information when code products or components are added or modified.

The configuration manager component provides the interface to create, change, or delete configuration objects automatically or under user control. These objects are used by programs to access and use hardware resources, such as printers, display stations, communications lines, tape units, and diskette units. Creation of configuration objects for directly attached resources is controlled by an automatic-configuration option. If the automatic-configuration IPL option is set to YES, configuration objects are automatically created for all locally attached devices when they are installed and powered on. If automatic configuration is set to NO, the user creates configuration objects using menus or commands. Default values are provided with OS/400 to minimize the amount of data that must be provided by the user.

The software vital product data and configuration information is used by the configuration manager component to correctly install and maintain all licensed program products and microcode groups. Product configuration and dependency-checking ensure program temporary fixes (PTFs) are correctly ordered and applied.

Problem Management Functions

New problem management functions are provided for managing system-detected problems. The AS/400 system also allows the user to manage problems identified and defined by the user (referred to as user-defined problems). Both of these types of problems are managed using the new problem management functions. The AS/400 system provides functions for automated problem analysis, automated problem logging and tracking, automated problem reporting, and problem correction. This is done to help the customer and IBM quickly and accurately resolve problems occurring on the customer's system. Figures 3 shows the AS/400 structured problem management facilities and their relationship to the support structures.

As an example, when problem management starts with a hardware error that is detected by a device attached to the system, an error notification is reported to the system using the common I/O architecture. The data pertaining to the problem contains vital product data and configuration information, the reference code and the name of the associated reference-code translate table for the reporting device, as well as additional failure information. The error is recorded in the system error log and an entry is created in the system problem log. A message is also generated and sent to the user. From this message, problem analysis can be entered for the specific condition and the analysis results can be automatically stored, information about the problem can be collected online, and the problem can be automatically reported to the IBM Service Support System for resolution. After the problem is reported, a service representative may be dispatched, a PTF may be sent to the system, or the appropriate support organization may be notified of the problem so they can contact the customer about it. The problem log is used to track the problem as it progresses to resolution. Alerts and alert management are provided to extend local problem management into the network problem management aspects of Communications and Systems Management (C & SM). The alert capabilities are flexible and allow various options for participating in network problem management.

Problem Log and Tracking

The problem log and problem log manager component are central to the AS/400 service philosophy. IBM service representatives, the IBM Service Support System, and the customer can use the problem log to determine the level of analysis performed and the locations for the hardware repairs associated with the problem, and to access any notes associated with the problem. The log can also be used to determine which PTFs exist that can correct the problem.

The problem log and problem log manager component provide a consistent means of tracking both user-defined and system-detected problems from initial definition through resolution. The tracking mechanisms indicate the state of the problem, including whether or not the problem has previously been analyzed. The problem log manager component provides the entry point for structured problem management on the system, guiding the user to the next step of problem resolution to be completed for a problem.

The problem log provides online tracking and data for each problem as it progresses to resolution. A problem record includes: the problem state; hardware and software vital product data; isolation at the time of failure and isolation after running problem analysis; replacement parts lists and part location information; service entitlement information; contact numbers; effect on customer operations; and actions taken as a result of reporting the problem through the IBM Service Support System. In addition, notes can be kept in the problem log for each individual problem and a problem can be updated by the user to indicate it has been resolved. The records are deleted at the user's request after they have been kept for at least 30 days. This time period helps to ensure that a problem can be adequately tracked.

Each problem is tracked independently through the different states. Options to run various problem management functions are provided by the problem log manager component based on the state of a problem. The problem state definitions and the selectable problem management functions for each state are:

- Opened: This is a new problem detected by the system. Problem analysis, problem reporting, and problem recovery functions can be selected for problems in this state.

- Ready: Analysis of the problem is complete. Problems at this state have either been detected by the system and analysis has been completed, or the user defined a problem to the system using the Analyze Problems (ANZPRB) command or function. Problem reporting and problem recovery functions can be selected for problems in this state.

- Prepared: Problem reporting has been selected and information has been collected for contacting IBM. Problem reporting, verification, and recovery functions can be selected for problems in this state.

- Sent/Answered: Problem reporting has been selected and a call sent and responded to from the IBM Service Support System. Resolution, verification, and recovery functions can be selected for problems in this state.

Problem Analysis

Structured problem analysis can be initiated for system-detected problems by running problem analysis, and for user-defined problems using the Analyze Problem (ANZPRB) command. Additional problem analysis is possible through non-directed use of system tools.

Problem analysis provides for the isolation and resolution of system-detected problems. It is designed based on the philosophy that isolation results are provided at the time an error is detected. This is done to reduce problem analysis effort and to enhance the accuracy of the analysis for system-detected problems. Significant errors result in system messages and are identified as a problem when the message is displayed to the user. Problem analysis may be started from the message that was displayed for a particular problem or from a list of problems displayed by the problem log manager component.

Problem analysis provides a common way to run resource-specific problem analysis procedures to analyze a specific problem. A component reference-code translate table is used to determine whether further analysis is needed for a particular problem or whether the analysis is complete at the time of failure. If further analysis is required, the component reference-code translate table is used to determine which problem analysis procedure to call initially for a problem. Problem analysis procedures are provided by components of the system for analyzing problems and verifying repair actions for problems. Problem analysis procedures and reference-code translate tables are supplied for hardware components, such as devices or adapters and certain software components.

Structurally, a problem analysis procedure is a program written in a special procedural language. The problem analysis component provides common services such as problem analysis procedure initiation and problem analysis procedure recovery. This structure isolates the problem analysis procedure from OS/400, allows the addition of new I/O, and provides a greater potential for using problem analysis procedures on other systems.

The Analyze Problems (ANZPRB) command can be used when the user discovers a problem that has not been detected by the system. The ANZPRB command guides the user through a series of panels designed to resolve user problems, isolate problems to a failing component, or generate a symptom list for reporting to IBM. During the definition of a user-defined problem, guidance is given to ensure that a procedural error was not made on the part of the user. Problem analysis procedures are supplied by system components as the entry points from the ANZPRB command. Once the problem is isolated to a component, the analyze problem function determines which general-entry problem analysis procedure to call. The function generates a symptom string for a software error, which is later used by the IBM Service Support System to determine if a software problem already has a PTF available.

In addition to structured problem analysis, the user also has the capability of doing non-directed problem analysis through access to the system's service functions. These can be called from the system menus, through the System Service Tools, through the Dedicated Service Tools, or by entering commands.

Automated Problem Reporting and Service Support

When a problem has been isolated through the structured problem analysis functions, the user is then given the option to report the problem to IBM. Selecting this option causes the AS/400 system to automatically initiate a communications session with the IBM Service Support System. The System Support Facility provided on the AS/400 system communicates, using the service call record interface, to software functions on the IBM Service Support System to perform service entitlement and call-record analysis, and to resolve or route the call. This system automatically communicates the requests through an LU type 6.2 program-to-program interface.

A service call record is passed to the IBM Service Support System for both hardware and software problems. For software problems, a symptom string is used to search an IBM Service Support

System data base to find available PTFs for this problem. If a match is found, the appropriate PTFs are sent to the user's system for subsequent application. For hardware problems, a call record is used to supply the IBM Service Support System with the failing-parts information and the IBM service representative is dispatched with the parts. If no hardware or software resolution is identified for a problem or the repair action requires onsite assistance, the call is automatically routed to the appropriate hardware or software support structure.

Copy Screen Image

The copy screen image function provides additional problem determination capabilities for the customer to debug applications, or as part of help-desk support. A value-added remarketer (VAR), value-added dealer, or third-party development support personnel can similarly use this function. Copy screen image also enables IBM support personnel to directly participate in problem determination on a customer's system.

Copy screen image provides the capability for a user at one work station to view the displays being viewed by a second user at another work station. Through commands or menus, the user specifies that display images on a specific work station are to be copied on some other work station. The display being copied is the controlling display station and is input-capable; it is called the source device. The display station to which the controlling display station's displays will be copied is a display-only device. It is called the target device and has no input capability.

The commands or menus used to control the copy screen image processing are very flexible. When starting the processing, the requester can identify, by device name, which work station is to be the source device and which the target device. Requesters can also specify that their device will be the source device, the target device, or not involved in the actual copy screen image operation. The processing can be ended by the user of the source device, or by a person not involved in the operation, using the command or menu interface. The user of the target device can also end processing through the use of a special system-request interrupt. The user also has the capability to direct copied display-images to a data base file. This enables the user to process this data at a later time and serves as an audit trail for what occurred during the operation.

Network Problem Management Support

The network problem management functions, combined with the enhanced problem management and configuration management functions, provide a comprehensive set of functions which allows management of the system within a network that works together with the System/370 C & SM functions. A problem is reported in the form of an alert, generated at the time of the failure, from the data available about the condition. The AS/400 system uses IBM's new generic alert architecture for the alert structure. The alert functions allow a high degree of customer selection of alertable conditions and provide detailed data for that alertable condition.

The AS/400 network problem management functions provide flexible control mechanisms. The customer can indicate that alerts are to be sent for all alertable conditions, for all alertable conditions except those defined as unattended, or not for any conditions. The customer can also control the sending of alerts on an individual-message basis by setting the Alert Indicator field in the message. This indicator can cause the alert to be sent immediately, after local problem analysis, only if the system is being operated in unattended mode, or not at all for a message.

The AS/400 system uses messages to identify alertable events and to provide part of the data needed for an alert. Other data that is provided as detailed alert data is contained in the reference-code translate table. The AS/400 system also supports generation of alerts from data entered by an operator.

The AS/400 system can be configured to operate in the following roles for receiving alerts:

- Alert Focal Point: Receive, log, and provide NetView™ Distribution Manager display capability. The configuration can optionally direct the system to forward alerts to a higher-level focal point. This is the hierarchical (or nested) focal point capability.

- Alert Forwarding: Receive and forward to a higher-level focal point with or without logging when the system is not configured as a focal point.

The AS/400 system uses a sphere-of-control table to send requests to be the focal point for those nodes capable of accepting the request. The sphere-of-control table allows the user to define all the network nodes that are to report to the focal point in a single table on the focal point system. The AS/400 system can be defined to operate as a primary or as a default focal point.

The AS/400 system receives both the new generic alerts and the stored-display alerts that use the major vector-subvector format. Alerts are received on either of the Systems Network Architecture (SNA) session types used for alerts. The AS/400 system will forward alerts when configured to do so. Alerts are forwarded on the appropriate SNA session, independent of the session type on which they were received.

Technical Support and Information Access

Technical support and information access, as an integrated part of OS/400, is new on the AS/400 system.

Technical support and information access is provided locally on the user's system and remotely through IBM-supplied marketing support systems (where a marketing support system is accessible). Figure 4 shows the relationship to the support systems of the three basic functions: question-and-answer, IBM product information, and technical information exchange (TIE).

The question-and-answer function is integrated within OS/400 and provides the user with a productivity tool for commonly asked questions and their answers on selected topics. The question-and-answer function is delivered with the operating system along with an initial IBM-supplied local question-and-answer data base. VARs and central-site support organizations can also use the question-and-answer function to supply their own question-and-answer data bases. IBM product information provides access to market support functions, such as system library subscription service lists (SLSS), announcement letters, and the like. Technical information exchange is an asynchronous file-transfer vehicle used to exchange files between the user's system and the IBM market support system.

Online Questions and Answers

The question-and-answer function provides the capability for a set of hierarchical data bases that users can access to improve education and technical information distribution. A local question-and-answer data base can contain commonly asked questions and their answers, with associated topics and search words that enable an end user to quickly find information. The question-and-answer function is divided into two primary areas, the set of operations and the question-and-answer data bases that can be operated on. The set of operations include question-and-answer data base management and item management within a question-and-answer data base. The question-and-answer function has the ability to manage a number of question-and-answer data bases that could include subjects

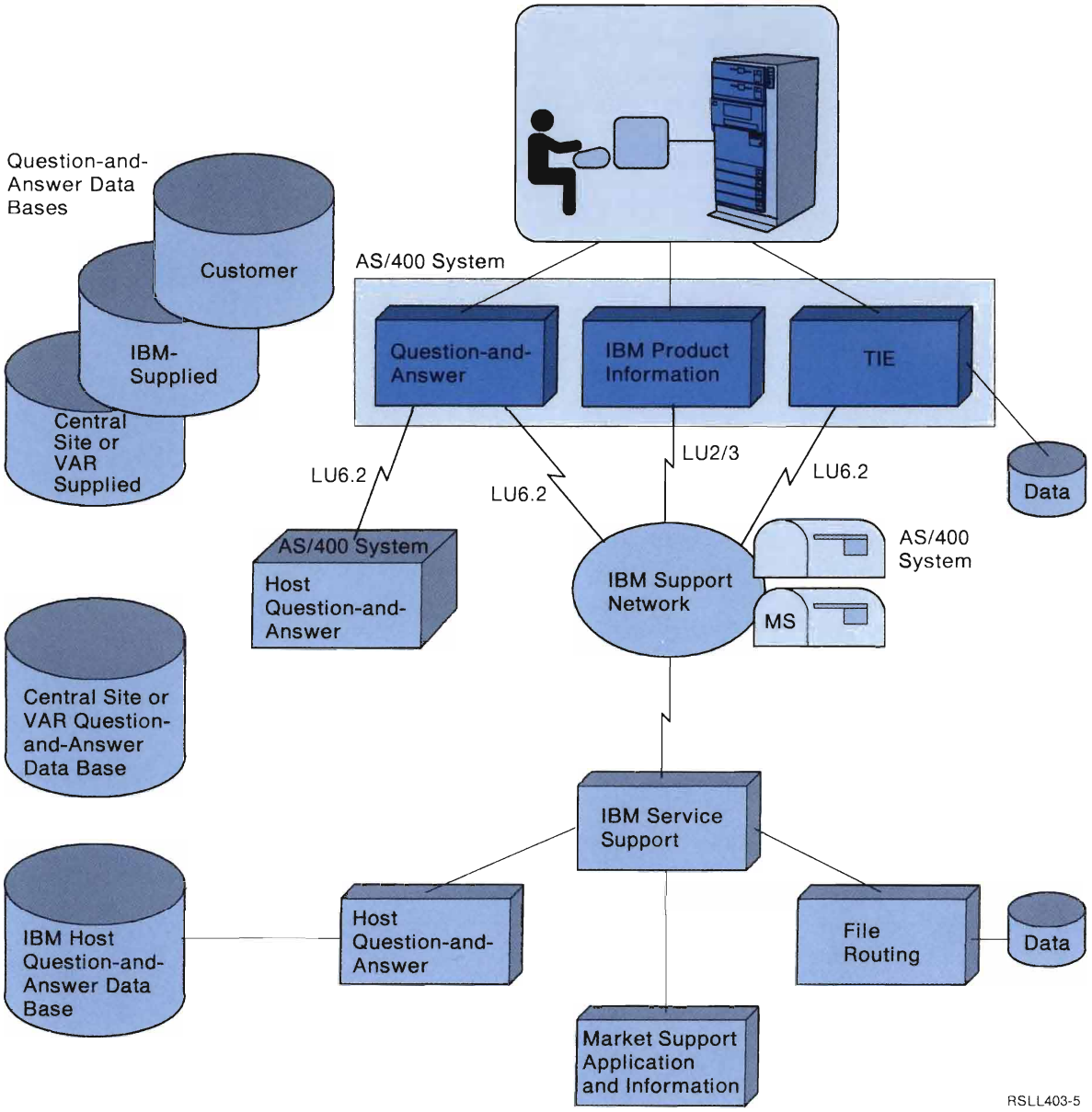


Figure 4 Technical Support and Information Access Functions

addressing technical topics, management or employee briefings for company policy changes, procedural directions, or operational questions. The topics are determined by the user or the data base supplier.

Each local data base can be associated with a remote data base. The remote data base can reside on an IBM host system (in one of several countries) or another AS/400 system. All of these remote data bases are accessed through the LU type 6.2 program-to-program interface, allowing the same local question-and-answer functions to be used for different host question-and-answer programs and data.

The question-and-answer function has three primary user types: the general user, the question-and-answer coordinator, and the administrator. These users are established through normal system security and may vary from question-and-answer data base to question-and-answer data base. The general user can perform searches on the local data base only. If the searches are unsuccessful, questions can be submitted to the coordinator for response. The coordinator can perform local searches, as well as searching the associated remote data base. The coordinator is also the responder for general user-submitted questions. The coordinator is considered a general user when accessing the associated remote data base. The administrator is responsible for the management or distribution of a data base. The administrator has all the rights of the general user and coordinator, plus data base management and distribution capabilities.

The question-and-answer function is easily accessed using system help commands or an easy-to-use set of menus. Local end users are shown the same style and basic content of displays, regardless of whether they are accessing local or remote data base information. The user interface is consistent with other user interfaces on the system, thus providing the functions in a familiar format.

Question-and-answer data bases can be supplied from several organizations. IBM ships with every entitled system an initial set of local questions and answers for use by the customer online. In addition, IBM provides access to an associated remote IBM data base. The IBM local and remote data bases place a wealth of knowledge and broad range of experiences at the fingertips of the AS/400 question-and-answer user. The customer can choose to supply a local data base, containing questions and answers that have been collected. This allows administrators to distribute commonly asked questions to their network of systems and, if desired, they can also be associated with a larger, company-wide data base.

The question-and-answer data base is divided into four primary areas: supplied, locally added, candidate, and conversational. When general searches are performed, the supplied and locally added portions of the data base are used. When questions are submitted, they are kept in the conversational portion of the data base. The candidate portion of the data base is used for the staged or controlled publication of items. This structure allows the user the freedom to tailor the searchable set of items in any one data base without affecting the supplier's information. The structure of the data base also allows the supplied portion of the data base to be refreshed without destroying any questions and answers that are being responded to (submitted questions), being published, or that have been locally published to all users.

The question-and-answer function also provides functions to create, distribute, and manage a new question-and-answer data base. Once created, the user can use the question-and-answer edit function to establish a base set of questions and maintain these items in the data base. After selected items are developed, the user can create a distribution copy of the data base from either the supplied portion, the locally added portion, or a combination of both of these portions. When this data base load is installed on another system, it provides that local user with a new supplied set of items to work from.

Questions and answers can be added to any local data base (including the IBM-supplied) at the customer's discretion. These items can be published for other users to access in their searches. Items can be published immediately or in a controlled fashion. Immediate publication of an item allows the user one chance to edit a question prior to publishing it. The less-formal user would likely publish questions and answers immediately when the general-user audience is smaller. The more sophisticated user (for example, a VAR or central-site user) would typically stage the publication of questions and answers to ensure accuracy, style, and content. These questions and answers are added to the locally added portion of the data base and do not affect the supplied portion provided by the supplier of the initial data base.

Marketing Technical Support

IBM product information and technical information exchange (TIE) work together to provide a set of technical support functions and information to the user.

IBM product information provides access to support tools that are available on the marketing support system. Examples of the available support tools include SLSS lists, AS/400 configurator, and access to marketing announcements material. Using the AS/400 3270 device emulation support, IBM product information provides the capability for the support system to use both display (LU2) and print (LU3) capability.

TIE provides two functions: the IBM Support Network connection support and an asynchronous file-transfer capability. Connection to the Support Network is used to establish an LU type 6.2 conversation with a partner application on the IBM Support Network. It provides the required network sign-on support. TIE asynchronous file transfer provides access to a marketing support mailbox in the IBM Support Network. Saved and open files can be sent to the AS/400 system from marketing support. Configuration and open files can be sent to marketing support by the AS/400 system using the information-exchange send and receive functions. Users can query their mailbox to determine if data is present.

Additional routing capability is added through file header information. This enables files to be sent to the AS/400 user's IBM product information marketing support system sign on or to the user's marketing support representative.

Conclusions

Designing system components for time-of-failure error detection and failure-cause analysis is a key design decision in meeting system support objectives. Implementing time-of-failure detection and analysis enables automated functions for isolating intermittent problems and improved accuracy for isolating other classes of problems.

Integrating the electronic customer support and C & SM functions into OS/400 and designing them for concurrent operation are also key to meeting system objectives. Concurrency of the functions improves system availability to the user. The problem management support in the system provides responsive support for problem isolation and reporting and provides immediate or deferred resolution of problems. Automated problem reporting and the user-defined problem resolution capability increase the effectiveness of personnel supporting system operations. The question-and-answer function and marketing technical support provide new levels of information access and exchange, enabling users to more efficiently learn and manage information about their systems.

The electronic customer support functions integrated into OS/400, and the interfaces they provide to other support functions and information outside the system, provide the user with a cohesive view of problem management and information access. These functions are available to the user in the language of choice at any display station attached to the AS/400 system.

The AS/400 system has made significant advances in the richness and ease of use of system management and support functions. Electronic customer support is the new standard for excellence in the computer industry and is expected to become the benchmark for all other systems. These functions are the foundation for future electronic customer support and C & SM enhancements at both a system and network level.

The System Capacity Planner

Describes the advanced capacity planning functions available as part of the AS/400 system.

Michael J. Denney, James M. Mickelson, and James C. Stewart

Introduction

Given the normal growth within a business and the ever-increasing application demands being put on the data processing area of a business, the need to plan for initial and changing system requirements is apparent. The overall goal for the AS/400™ capacity planner (the Model System command, MDLSYS) is to provide an easy-to-use tool that can assist with these tasks.

The AS/400 capacity planner represents unique approaches in the area of capacity planning. The capacity planner is made up of five major components. The work load component assists the user in defining and characterizing the applications running on the system. The configuration component validates the system configuration, while the analytic model component uses the work load and configuration information to predict end-user response times, throughput, and system utilizations. The evaluator component analyzes the model predictions and recommends a configuration change to improve the response time, throughput, and utilization levels based on the user's objectives and design guidelines. The growth component allows the user to analyze future system requirements based on annual growth rates. These five components work together to provide an easy-to-use capacity planner.

Work Load Component

The work load component allows the user to characterize the application work load. Because system resource requirements are generally tied to specific components of a business, the total system work load is treated as a collection of measurement profiles that correspond to the various business components. On an installed AS/400 system, this data input is automated through a measurement profile generated using the Print System Report (PRTSYSRPT) command. The PRTSYSRPT command is available as part of the AS/400 Performance Tools.

Figure 1 illustrates generating measurement profiles for the three business components accounting, sales, and online ordering. Within the capacity planner, the measurement profiles are combined to give a total system view of the work load. This combined set of measurement profiles is referred to as a response file.

This ability to characterize work being done on the system at the business component level has a big advantage. Growth within a given business component while holding other components constant can be examined. New applications that have been characterized in the form of a measurement profile can be added using the

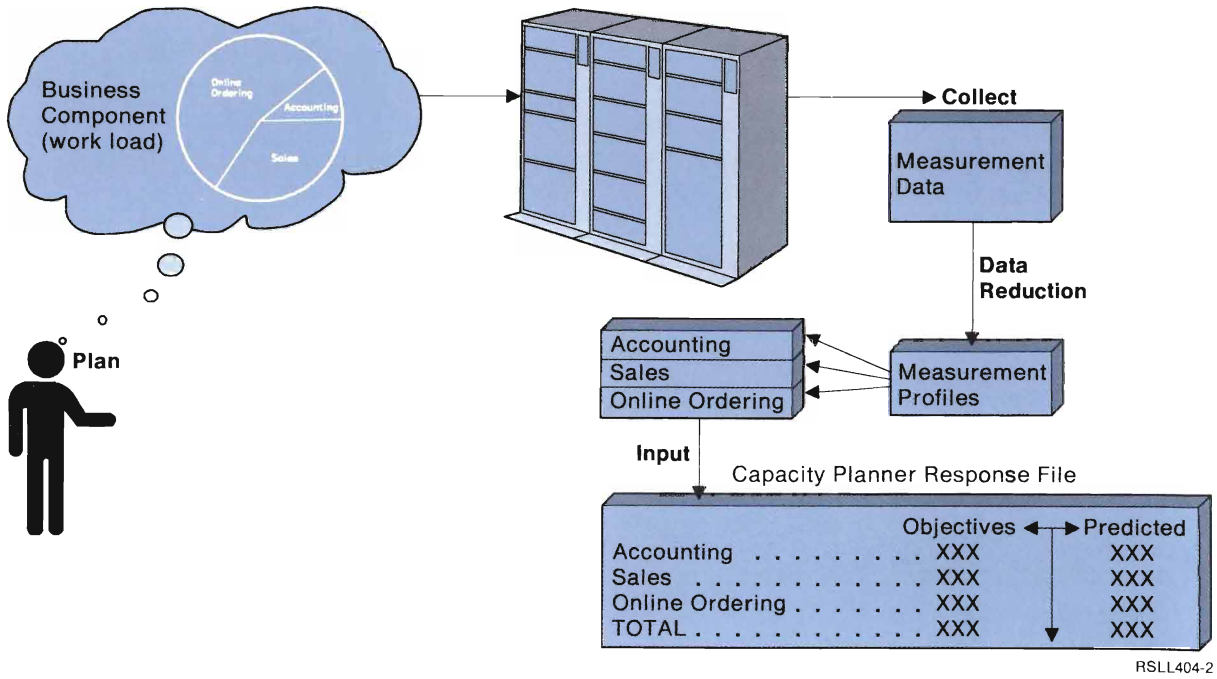


Figure 1 Measurement Profiles as Input to Capacity Planner

capacity planner to see their effect on the system. Also the user can add other types of work to this, such as the RAMP-C™ benchmark program, batch, and spool profiles. Objectives for throughput (transactions per hour), response time, and active work stations can also be specified as input to the capacity planner.

Configuration Component

The configuration component ensures that the capacity planner analyzes only valid combinations of processor models, main storage, disk devices, and communications devices. This helps the user order hardware upgrades. In a proposal situation (the user does not have an installed system to measure), the configuration component's first job is to calculate an initial configuration: a starting point to work from. This initial configuration is then analyzed by the capacity planner and is modified until the user's objectives are met.

To calculate the initial configuration, the configuration component looks at the user's throughput objectives and the application characteristics (number of instructions, disk accesses, working-set size) and applies queueing theory to calculate the number of devices required to keep the utilization within recommended levels. (The working-set size is the main storage requirements for a job and includes all of the objects necessary for a job to process.) From this point on, this component's job is to ensure that any configuration changes result in valid system combinations and to know the proper hardware needed to upgrade each device.

Analytic Model Component

The analytic model component predicts the response times, throughputs, and utilizations based on the work load and configuration components' output. Because the model processes quickly, it can produce a range of performance predictions for the current configuration by repeatedly increasing the number of active work stations and analyzing the results. The user can then see, at a glance, what the predicted performance will be as the system gets busier.

The queueing model for the capacity planner is comprised of several submodels. These submodels, with one exception, use an approximation of the mean value analysis. Resources modeled using this technique include the System Processor, the disk subsystem, activity levels within the interactive storage pool, the work station controllers (local and remote), and the remote lines. The exception to the mean value analysis is the model that predicts the number of disk accesses per transaction, referred to as the paging model. This model is essential to predict the amount of main storage necessary to accommodate all of the active jobs.

To model paging, many measurements were reduced into paging curves, where the overcommitment ratio of main storage is the X axis, and the number of disk accesses per transaction is the Y axis. The overcommitment ratio is calculated based on the number of active jobs, the jobs' estimated working-set sizes, and the size of the main storage pool where the jobs run. Thus, the overcommitment ratio is a ratio of the amount of main storage to contain all of the active jobs' objects compared to the amount of main storage available in the storage pool.

The pre-characterized work load in the capacity planner (RAMP-C) has a paging curve. The measured profile input by the user, however, does not have a paging curve; this profile is only one point on a paging curve. To produce a paging curve for a measured profile, the RAMP-C paging curve is adjusted to pass through the measured profile's one point. The curve is then adjusted based on the difference between the estimated working-set size of the measured profile and the estimated working-set size of RAMP-C (see Figure 2).

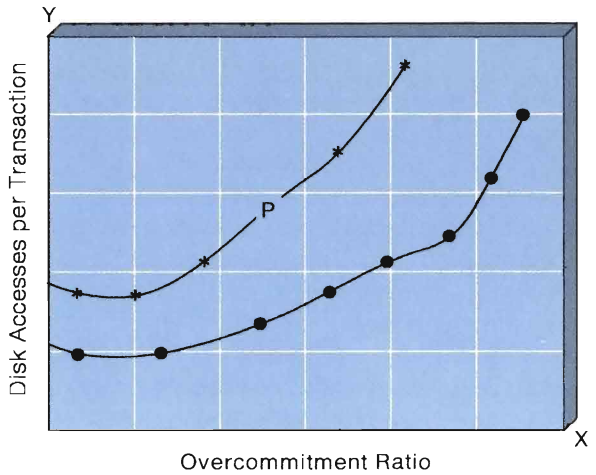


Figure 2 Measured Profile Paging Curve

Evaluator Component

The evaluator component provides the expertise in analyzing the output of the model. The evaluator knows when system performance is unacceptable or close to being unacceptable. For example, if the interactive portion of the processor utilization is 75% (the processor is busy 75% of the time), performance may still be acceptable. However, the system does not have much reserve capacity, and performance will degrade rapidly if this utilization goes up as a result of adding more work load. In this situation, the evaluator will recommend a faster processor.

However, some performance problems are not as easy to identify. For example, a high disk utilization may have multiple causes, such as: excessive paging due to insufficient main storage; high disk to input/output (I/O) processor utilization; or high disk to controller (A-Box) utilization. The evaluator can identify each of these and recommend a change to fix each problem. The evaluator also tells the user if the change is

required or optional. Required means that utilizations are above recommended levels; optional means that utilizations are approaching these levels and the user should be concerned. The values for these levels are set to reflect the performance observed in benchmark measurements, provide reasonable and consistent average response times, leave expansion room for future growth, and account for variations in the work load on the system throughout the day.

When the evaluator has identified the problem and recommended a change, the user can either accept the recommendation or make a different change to the configuration. In either case, the configuration component will validate the change, the analytic model will predict the performance, and the evaluator will analyze the results again. This process continues until the user is satisfied with the configuration.

The evaluator was originally implemented using expert system tools and techniques. A typical expert system consists of a rule base and an inference engine. The rule base contains the domain-specific knowledge in the form of IF condition, THEN action statements, called rules. The inference engine decides which rules should run based on the available data.

The evaluator consisted of a set of performance rules that were easily implemented with the rule base and inference engine approach that expert systems use. The expert system approach allowed the rule base to be easily modified as more rules were needed or if the rules needed to be reorganized. The developers could concentrate on perfecting the performance rules and did not have to be concerned with program flow or where each rule should be located; the inference engine took care of that.

The evaluator was eventually translated into a conventional program because Operating System/400™ (OS/400™) currently does not have any expert system support. Also, because the capacity planner is available on systems using Virtual Machine (VM), a completely different operating system, a high-level language that is supported by both operating systems was necessary.

Growth Component

When the system configuration has been determined, growth predictions can be done to determine the long-range data processing equipment needs. The user could accomplish this function by repeating the entire capacity planner procedure for all periods of interest (one, two, and three years from now, for example) by calculating the number of active work stations for each period, based on the expected growth rate, and creating the input for each analysis. This could be a time-consuming process.

However, the capacity planner allows the user to specify a growth rate for each application with three time periods to analyze. The growth component calculates the number of active work stations for each period and then uses the model, evaluator, and configuration components to find a system configuration that will handle the additional work. Each configuration will have utilizations below the recommended levels and response times that are below the user's objectives.

Conclusions

The AS/400 capacity planning function provides AS/400 customers with a way of planning for their initial and future requirements. Its integration into the AS/400 Performance Tools provides a measurement interface and a level of analysis that greatly enhance its function and usability. The many variables that need to be considered to arrive at a balanced system configuration are handled by the capacity planner's analysis. The sophistication of its decision-making process assists in solving the very complex problem of data processing equipment planning.

Software Design to Support National Languages

Describes software packaging based on the physical separation of textual data from operational program code using a building-block design, allowing licensed programs to be distributed and installed in multiple national languages.

Eric L. Fosdick and Michael F. Moriarty

Introduction

IBM designs software products that allow user interaction in the national language or languages chosen by the user. The textual data displayed or printed by a software product is available in many national languages, such as English, French, German, and Japanese.

This textual data consists of messages, prompts, displays, and online documentation. A software product that contains textual data in a specific national language is called a national language version.

Typically, the process of packaging, testing, and distributing multiple national language versions for many licensed programs was very involved. Each national language version was created at a different location around the world in a process that was time-consuming and difficult to control. This process also makes the capability of having a concurrent, worldwide availability date for all national language versions very difficult, if not impossible, to achieve.

The design of the AS/400™ software separates textual data from operational program code when it is packaged on the distribution media. This separation is achieved using a building-block concept for software packaging.

The AS/400 design provides several positive results. First, the process makes concurrent worldwide availability of licensed programs in multiple national languages practical. Textual data for licensed programs is translated into multiple national languages, tested, and packaged independently from code. In addition, this method of software packaging supports multiple national languages simultaneously available on one system. The user can select one or more national languages when installing software products. And finally, the textual data can be updated by national language between software releases, thereby giving the system user more timely support.

Separating Textual Data From Operational Program Code

Previously, textual data was designed to be separated from the operational program code as unique objects to allow for text translation from English to each supported national language. Creating national language versions of the licensed program involved two basic steps: first, the English text was translated into a specific national language; and second, the translated textual data was integrated with the operational program code to create a national language software package for each licensed program.

The AS/400 design does separate textual data from operational program code to allow for national language translation. But, this design continues the separation of textual data from operational program code when they are packaged on the software distribution media. This allows for a software packaging methodology that uses a building-block concept [1].

For each licensed program, two separate building blocks are used for software packaging: the code building block and the textual data building block. The code building block contains all of the code objects for a specific licensed program, while the textual data building block contains all of the textual data objects for a specific licensed program. Each language has a separate textual data building block.

The operational program code and textual data building blocks are sent to a software distribution center, where they are packaged as separate entities on a customized tape that is sent to a customer. The operational program code and textual data are finally integrated into a common library during the licensed program installation process on the AS/400 system.

To fully support national languages, some licensed program code is national language dependent. An example is an operational-program code-page transformation table used to convert graphic characters from one operational-program code page to another. (This type of code is called national language dependent function.) The AS/400 design supports the packaging of national language dependent function in either the operational program code building block or the textual data building block for a specific licensed program. This packaging choice is determined individually for each national language dependent function.

The AS/400 building-block design and the resulting national language version of the licensed-program build process improves the capability of having a concurrent worldwide availability of all software products. This design

allows the process of creating national language versions of the licensed program at the translation centers to consist of translating, packaging, and testing the textual data objects only. This is simpler and less time-consuming than creating national language versions with both textual data and operational program code integrated.

Distributing to a Worldwide Audience

A primary result of the software building-block design is a better methodology for distributing software products to a worldwide audience. AS/400 software is distributed to the customer using a customized software tape containing the operational program code and primary national language textual data for each licensed program ordered. An additional tape is sent for each secondary national language that is ordered. The secondary national language tape contains the translated textual data for all licensed programs. It does not contain any operational program code.

The process of creating a customized software tape for a customer order is: (see Figure 1)

- The development laboratory sends the completed operational program code and the English textual data to the software distribution center. It also sends the English textual data to the translation centers.
- Each translation center sends the completed national language textual data for each licensed program to the proper software distribution center.
- The software distribution center creates a customized software tape using the corresponding operational program code and national language textual data that matches the customer order. The distribution center also creates a separate tape for each secondary national language ordered.

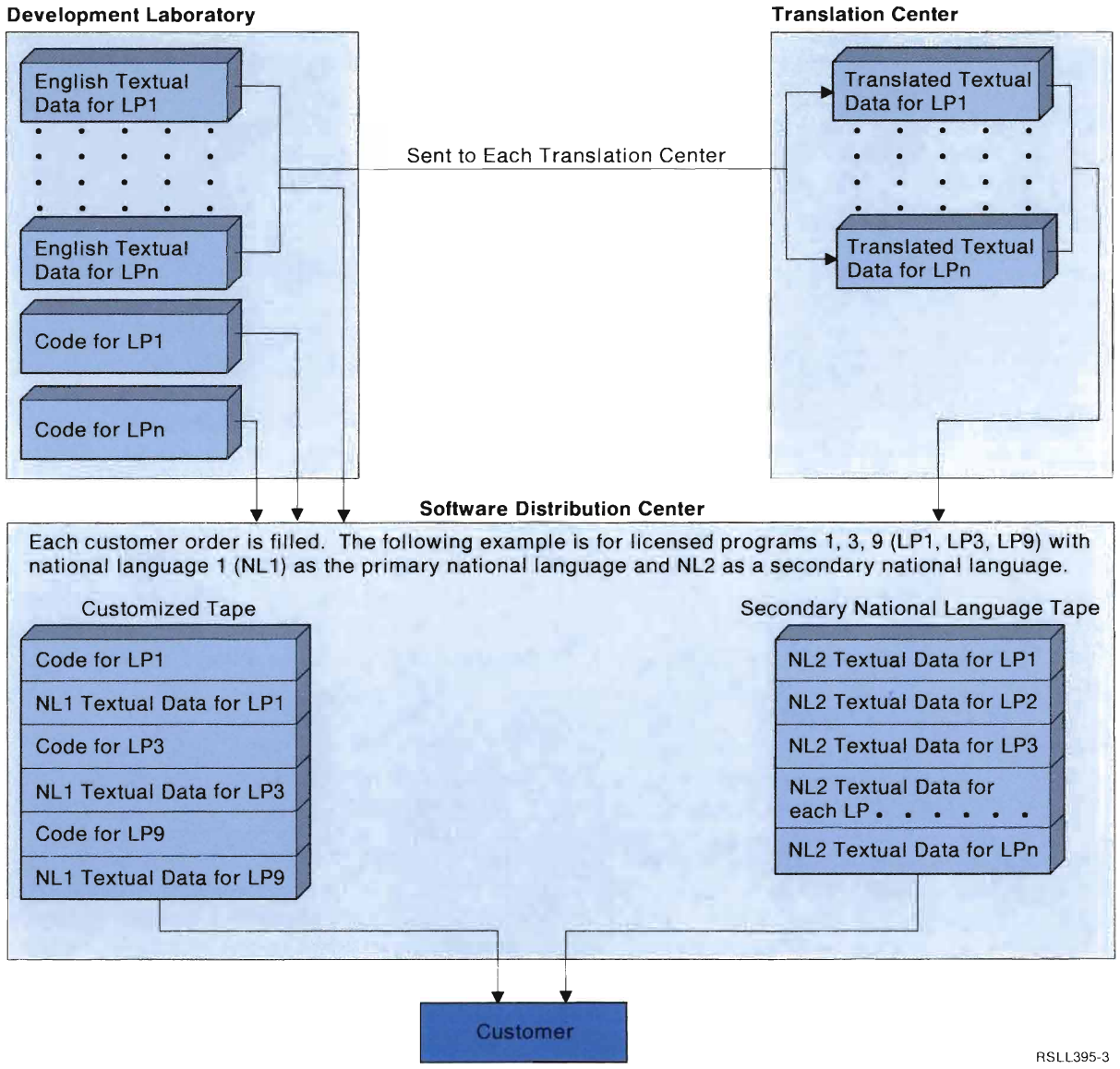


Figure 1 Process Flow Using Software Building Blocks

Although the current software distribution methodology for the AS/400 system uses a customized tape, the building-block design can be used to support other software distribution methods, such as all available software products packaged together on the distribution media.

Another result of the software building-block design is the capability to update national language textual data independent of scheduled updates or new releases for licensed programs. This capability provides the translation centers with the flexibility of staging the textual data translation between product releases. This is very important when high volumes of textual data are being translated.

National language textual-data updates can be created whenever the translation centers send updated national language licensed-program textual-data tapes to the software distribution center. The distribution center then builds and distributes an updated national language textual data tape to the affected customers without having to redistribute a customized software tape containing operational program code and textual data. This updated national language textual data tape is built the same way that a secondary national language textual data tape is built.

Installing the Software

The software is installed by reading the operational program code and textual data from the distribution tape and writing (restoring) them to libraries on the system. The software is installed in two phases.

In the first phase, the initial program load (IPL) process installs the vertical microcode (VMC) and the operating system. For the first system IPL, the IPL prompt contains the IBM logo and a single input field, into which the user selects the desired primary national language. This prompt is designed so it does not require translation. The tape also contains translated versions of the remaining IPL displays in each national language. The IPL process, however, exposes the user only to displays in the national language specified on the first prompt. If the language selected is not on the customized tape, the IPL process asks the user to insert a secondary national language tape. When IPL completes, the operating system is started and uses the language selected by the customer.

In phase two, operating system commands and menus are used to install the optional parts of the operating system and the licensed programs. The primary language for these is the same as selected in phase one. The operating system also supports multiple secondary languages, and the textual data for each is installed into its own unique language library. A menu interface guides the user in selecting one or more secondary languages.

Just as the textual data for the primary national language was initially installed in two phases, it is also updated in two phases. During phase one, the user specifies on an IPL prompt that *only* the textual data is to be updated. The IPL process stops at the appropriate time and requests a new language tape be inserted. The textual data on the new tape replaces the old textual data for the VMC and the operating system. In phase two, the system menus allow the user to update the primary national language textual data for the optional parts of the operating system and the other licensed programs. The menus also allow the user to update the secondary national language textual data.

Conclusions

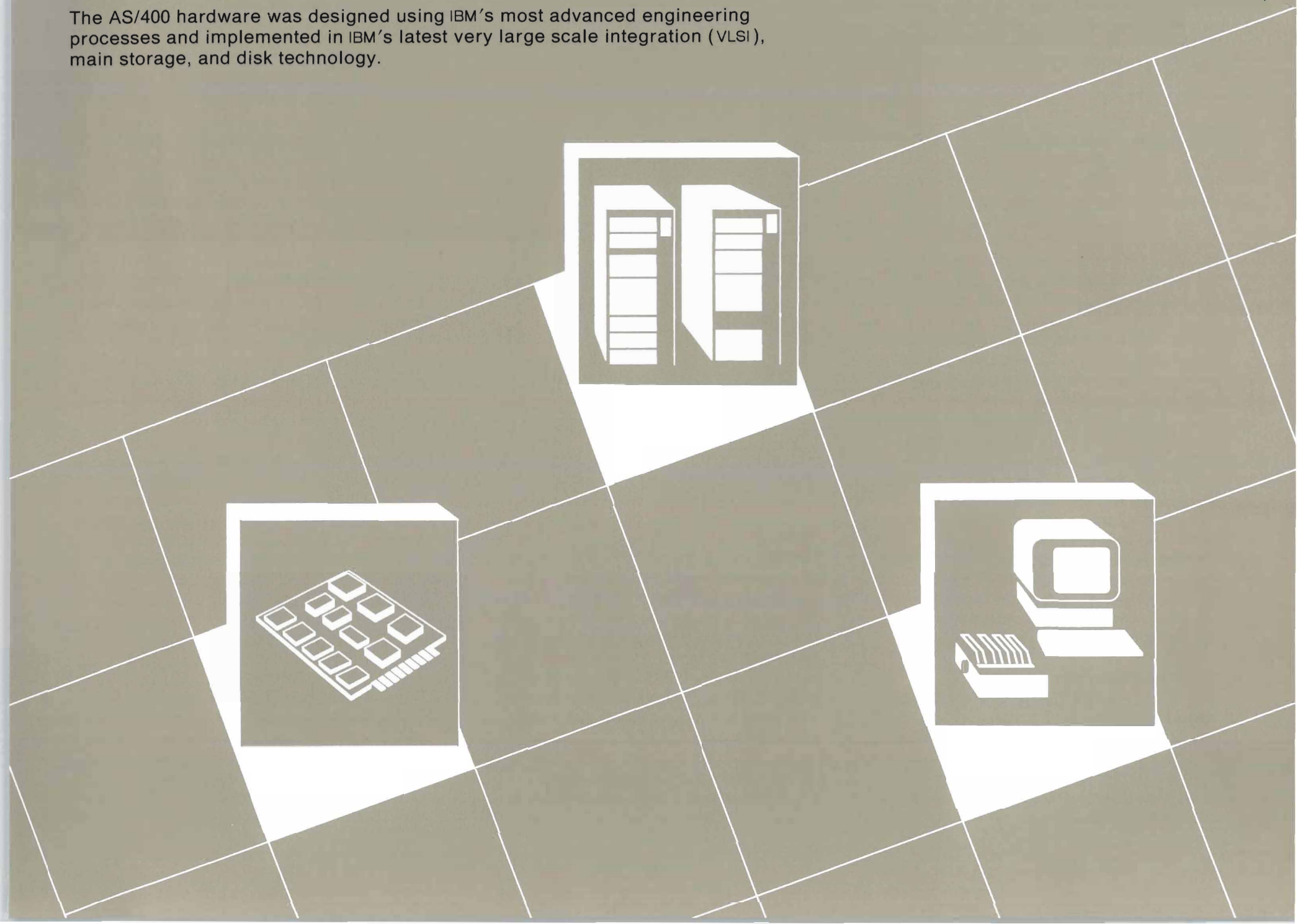
AS/400 software was designed from the outset to support a worldwide audience with many different national languages. Translatable textual data is physically separated from the operational program code until the user installs it. This separation enables the user to install updates to the textual data and to install additional national languages independent of the operational programs. Usability is enhanced by extensive prompts and menus that guide the user through the installation process. Another aspect of the design is that textual data and operational program code are packaged in building blocks that can be rearranged to meet future packaging, distribution, and installation processes in response to new technology and customer needs. These advanced features demonstrate a significant improvement in national language technology.

References

1. **National Language Information and Design Guide Volume 1: Designing Enabled Products, Rules and Guidelines,** SE09-8001. September, 1987.

Engineering

The AS/400 hardware was designed using IBM's most advanced engineering processes and implemented in IBM's latest very large scale integration (VLSI), main storage, and disk technology.



System Processor Architecture

Describes the layered machine interface developed for the AS/400 System Processors, which provides for enhanced system function and performance.

Mark R. Funk, Quentin G. Schmierer, and Dale J. Thomforde

Introduction

A primary AS/400™ characteristic is the unique high-level machine interface (MI). The machine interface separates the application programmer from the actual AS/400 hardware implementation and is the lowest-level instruction set available to the user. The instruction set used by Operating System/400™ (OS/400™) and high-level languages is also defined by MI instructions. As a result, MI allows programming independence from machine implementation and configuration details. Many of the basic supervisory and resource management functions of the operating system are implemented within MI. The actual support for MI is distributed through two internal microprogramming levels, vertical microcode (VMC) and horizontal microcode (HMC), and physical hardware. In Figure 1, the variable depth of each layer represents the distribution, or amount of support, of any MI function performed within that layer. The characteristics of the multilayer architecture and the flexibility of the internal interfaces allow function to be moved to lower levels of the machine in an evolutionary manner. This movement provides enhanced system function and performance.

The Machine Interface

MI is supported by a microprogramming level called vertical microcode (VMC), which is separated into two distinct classes of support. One class is the operating system, including such functions as storage management, data base management, and input/output (I/O) support. The second class is the translator, which converts

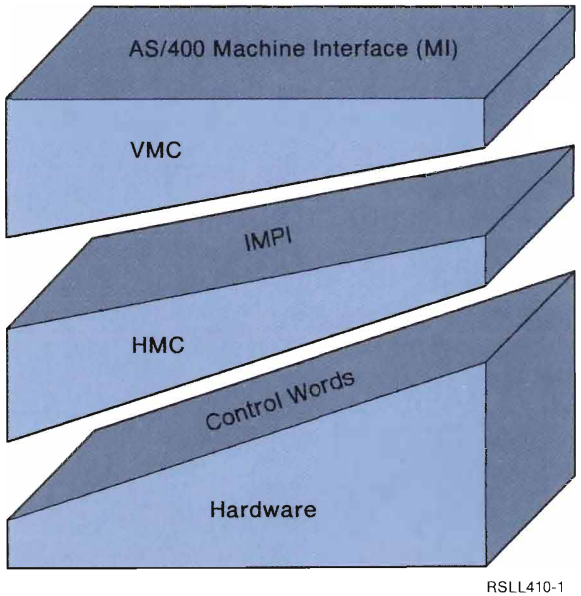


Figure 1 AS/400 Architecture

machine instructions into instructions at the internal microprogramming interface (IMPI) level. The conversion supported by the translator can be visualized as a compiler step. Individual MI instructions are converted into one or more sequential IMPI instructions, or into calls to VMC routines. The VMC routines themselves consist of IMPI instructions that implement the requested function.

IMPI also consists of two distinct classes of support. One class, as with VMC, pertains to operating system support. Within this class are instructions that do such diverse operating system functions as storage management, security and system integrity, data base management, task dispatching, task and message queueing, and I/O processing. The second class consists of machine instructions and extended-function IMPI instructions. IMPI instructions are interpreted by the next lower level of microprogramming, called horizontal microcode (HMC). The interpretation is supported by HMC routines, consisting of one or more HMC instructions called control words. The hardware directly decodes and processes the HMC control words.

AS/400 System Processor Features

The basic operations within the IMPI instruction set consist of a set of machine instructions similar to System/370 instructions, including register-to-register (RR), register-immediate (RI), register-and-storage (RS), storage-immediate (SI), storage-storage (SS), and branch-type instructions.

The register set in the System Processor consists of sixteen 48-bit base registers. These base registers are accessed in 8-, 16-, 32- and 48-bit mode by the RR, RI, and RS instructions.

The SI instructions typically process 8, 16, or 32 bits of storage against an immediate field in the instruction. The SS instructions process variable-length strings of characters and signed-binary or packed-decimal integers with a single instruction. Also at the IMPI level are instructions supporting IEEE floating-point, zoned-data (unpacked

main and disk storage. Instructions within this class have been found to have a relatively high frequency of use. As such, the functions they support were excellent candidates for moving into the hardware. These instructions also support the conversion between the 64-bit MI pointer address and the 48-bit virtual address at the IMPI level.

The System Processor includes many advanced data base management instructions. They support higher-level instructions used at the machine interface and in VMC. For example, the System Processor hold-free mechanism is used by the VMC seize-release routines and by MI lock-unlock instructions. This mechanism is implemented by a group of IMPI instructions supporting chained-hold records. The hold records represent lock or seize activity for a given system or data base object by all active processes.

IMPI task-dispatching instructions provide task or context switching from one procedure in a given task to a procedure in a different task. Queueing instructions support exchanging information and synchronizing the flow of control between tasks. The synchronization is provided through a send-and-receive message approach. I/O processing support is closely coupled to the built-in functions of queueing and task dispatching. The System Processor contains additional instructions that support subscript address generation for arrays, stack-space maintenance, various modes of context switching, and timers.

The System Processor 6-byte virtual address allows addressability to any byte within a 281-thousand gigabyte address space. The single-level storage aspect of the AS/400 system is supported using this virtual address scheme. Most storage references are made through a 6-byte virtual address generated through a base register plus displacement calculation. Any of the 16 System Processor base registers can be used in this manner.

decimal), and conversions between data formats. The large set of primitive operations allows generating compact and efficient code with short functional path lengths.

Of particular interest within the branch-type IMPI instructions are the composite, conditional test-branch, and compare-branch class. This frequently used class of IMPI instructions is translated one-for-one from a similar class of MI instructions. This is an excellent example of function that has been moved in an evolutionary manner from VMC to HMC, and then into the hardware with no impact to high-level applications. The functions supported by the branch-type class of instructions include the basic conditional branches, indirect and indexed branches, internal and external routine calls, function calls, supervisor calls, and associated returns.

In addition, the System Processor supports the more complex operating system functions of storage management, security and system integrity, data base management, task dispatching, queueing, and I/O processing. The storage management instruction class includes instructions that range from the simple translation of virtual addresses to the more complex determination of appropriate candidates for purging pages from main storage. The instructions in this class perform functions associated with the primary directory, which is the primary translation table between the IMPI 6-byte virtual address and the physical main storage address.

The security and system integrity class of instructions includes IMPI instructions that process and verify MI pointer objects. The MI pointers support 64-bit virtual addresses. An MI pointer is an object that is used only for addressing and does not permit examination or manipulation of the implied physical address. The validity of the pointers is assured by including a special tag bit in

Horizontal Microcode Features

The processor hardware does not process IMPI instructions directly. The IMPI instructions are converted into a series of sequentially processed HMC control words. The control words are directly decoded and run by the System Processor. In general, one HMC control word is run per processor cycle. For the lower-level IMPI instructions, a single control word, thus a single processor cycle, is required. More-complex System Processor functions are supported by multiple controls words and take proportionally longer to process.

Each HMC control word is 42 bits in length and is encoded into one of 13 different formats. The control word is subdivided into a variable-length opcode and a number of fields that are used to control the processor hardware. The fields control functions such as register gating, the arithmetic logic unit (ALU) operations, virtual address translation, memory accesses, and generation of the address for the next control word to be processed.

The position, size, and content of the control word fields were chosen to make maximum use of the System Processor hardware. Performance was enhanced by allowing parallel processing of important functions. Control words that take more than a single processor cycle to run are buffered to allow the Processor to continue fetching new control words without interference. With a single control word it is possible to add a displacement to a virtual address, translate the address, and initiate a memory access while moving data between other registers in the processor. An ALU operation could take place at the same time as a data move between registers and a memory-access request. System Processor status controls and generation of the next control word address are done in parallel with each control word. In addition to the synchronous parallel operations within the System Processor, which are under direct control of the HMC, many

asynchronous operations can also be initiated by the HMC or by the hardware, and are processed in parallel with an HMC control word.

High-speed random access memory (RAM) on the System Processor card is used to store control words. It contains either 8192 or 4096 locations, depending on the system model. Not all of the control words needed for System Processor support fit into control storage. Infrequently used HMC resides in main storage and is automatically retrieved by the Processor into reserved locations in control storage when it is needed. Control storage is loaded during initial microprogram load (IMPL) and is another example of system flexibility. Enhancements or new functions supported by HMC may be installed without any hardware changes.

Hardware Features

The System Processor cycle time is between 60 and 120 nanoseconds, depending on the system model. Most HMC control words run in one cycle. The System Processor is partitioned into six independent functional units. On the higher performance models of the system, the six functional units are implemented in six modules, with one chip per module (see the card on the right in Figure 2). Each high-performance bipolar chip contains the equivalent of up to 14,000 2-input NAND gates. Up to 240 functional I/O pins reside on each module. Other models package the six functional units into three single-chip modules (see the card on the left in Figure 2). Each module contains a CMOS chip with the equivalent of up to 40,000 2-input NAND gates and a maximum of 231 functional I/O pins. The partitioning of the functional units across the chips maximizes parallel operations while minimizing chip-to-chip signal crossings. The control word formats were designed to match the hardware partition. (For more information, see the article *System Processor Technology*.)

Two of the functional units provide main storage control. One of the storage-control functional units provides address and control for the storage cards. Each storage access can take from one to three cycles. Three independent address busses allow interleaved accesses across the address space. Up to 96 megabytes of main storage is supported. The functional unit also provides refresh control and storage-card control lines. Up to six storage cards, two on each address bus, are supported. The other storage-control functional unit provides error checking and correction (ECC) for data fetched from main storage. It also provides an interface for I/O storage accesses, which are interleaved with System Processor storage accesses. Data is written to and read from main storage across an 80-bit data bus. This includes 8 bytes of data, 14 bits of ECC check bits (used for error checking and correction of data, and error checking on main storage addresses), and a single tag bit (used for marking system pointers). The ECC algorithm used is capable of detecting and correcting single and double 4-bit package errors.

HMC control words directly control the remaining four processor functional units. Three of the processor functional units contain ALUs that are under HMC control. The hardware supports 8-, 16-, and 32-bit ALU operations. One of the functional

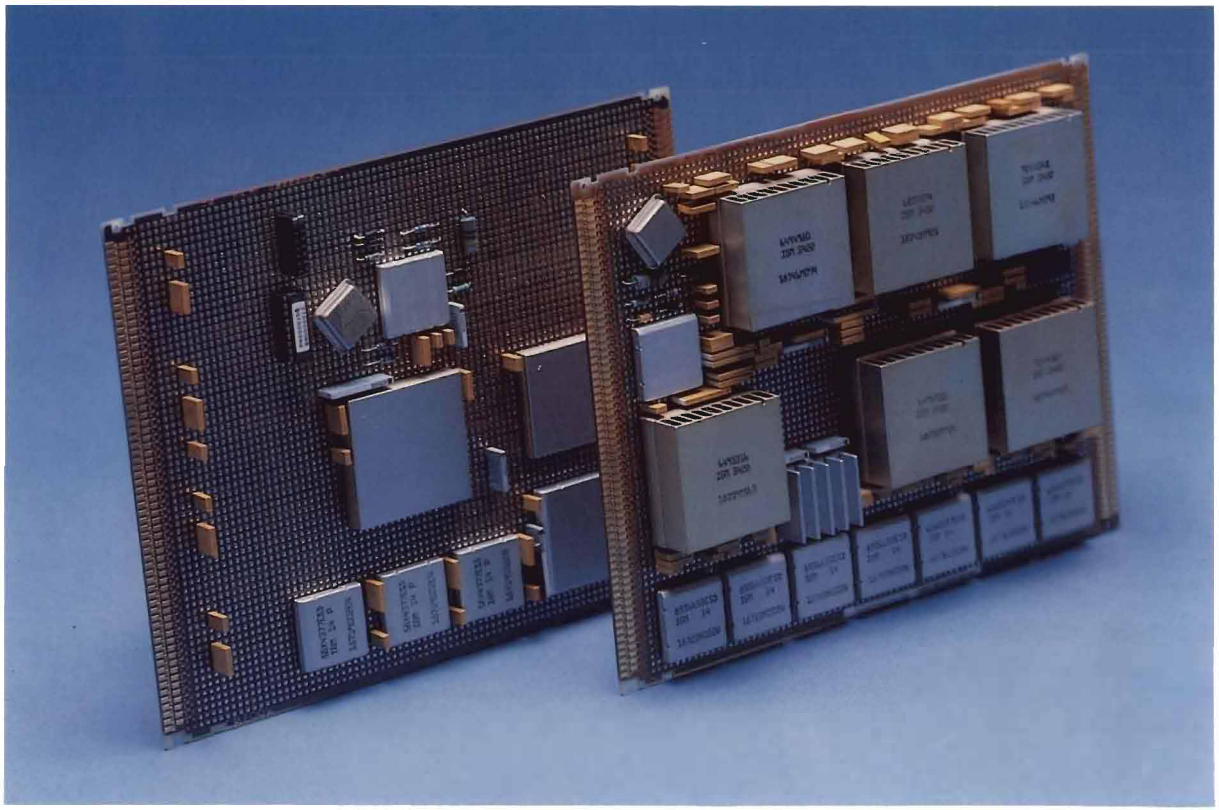


Figure 2 AS/400 CMOS and Bipolar Processor Cards

units supports pre-fetching IMPI instructions from main storage. An instruction pre-fetch buffer is used to reduce the amount of time spent waiting for the next instruction. It also contains a 16-bit ALU to process IMPI instructions, which modify base registers and calculate effective addresses. Another functional unit was designed to support storage, shift, and multibyte string instructions.

The third processor functional unit generates the control storage addresses and fetches control words. HMC conditional branching, branch and link, control storage overlaying, processor exceptions, and interrupts are supported by this unit. It contains an 8-bit ALU and a fast array that are used by HMC for process control information.

IMPI instructions address operands through a virtual address. Effective addresses are generated for IMPI instructions through a base register plus displacement calculation. The resulting virtual address must be translated into physical main-storage addresses prior to initiating a main-storage access. The fourth processor functional unit is responsible for this address translation. The translation hashes the virtual address to generate an address into a high-speed translation look-aside buffer. The look-aside buffer contains the most recently translated virtual addresses. The probability is better than 99% that the new address can be translated through one of the 1024 entries in the look-aside buffer. If the look-aside buffer translation was not successful, the hardware attempts to translate the address through the primary directory located in main storage. The primary directory is a table listing all virtual pages currently residing in main storage. Virtual addresses that are translated successfully through the primary directory are placed in the look-aside buffer with the corresponding physical page address. If the translation through the primary directory is unsuccessful, the status of the System Processor is saved and the virtual address being translated is passed to VMC, which will then copy a 512-byte page of data, starting at the requested address, from disk storage.

Conclusions

Given the high-level function supported by the AS/400 System Processor, the classical concept of the performance of a processor (instructions per second) becomes less descriptive. Such factors as the processor cycle time, the main storage data-bus width, the instruction pre-fetching, the 48-bit registers, and the many single-cycle IMPI instructions are indicative of the power of the AS/400 hardware. However, the performance of the System Processor is more than these hardware-related factors alone. The high-level System Processor function, which is efficiently supported by HMC (often with the appropriate hardware assistance), is indicative of the System Processor's performance.

Further, IMPI and HMC do not limit the growth of IMPI hardware. In fact, they are tools to achieve still greater levels of performance. By enhancing HMC and hardware, the System Processor's performance is increased beyond what could be achieved with hardware alone. All processor functions benefit, whether basic or high-level. Because of the flexibility, new function can be added to the System Processor.

As VMC supports the machine interface, it can also make full use of the enhanced System Processor, whether the function is provided by HMC or by hardware. Enhanced system function, with associated improvements to the machine interface, can be distributed through the levels of support best suited for the function. Additional performance improvements can be achieved at each level. Performance in such an environment (in terms of throughput, response time, and other high-level methods of measuring performance) improves at a greater rate and with greater potential than cycle-time related measures of performance.

The layered machine interface support allows remarkable flexibility and optimization within the supporting layers. As processor technologies improve in speed and density, a highly used function at the processor level can be moved closer to the hardware level by enhancing the HMC control words and underlying hardware. This allows greater performance with no impact to programs written at a high level. Also with this method, processors at varying levels of sophistication and resulting cost can support a constant level of IMPI through the appropriate HMC support.

In like manner, VMC routines deemed appropriate due to performance considerations can be moved into HMC by enhancing IMPI. New IMPI instructions can be added and others deleted. Because none of these things affect the machine interface, the user is never affected. New AS/400 functions can be partitioned into the various layers of the machine, as appropriate. This flexibility has allowed function to move to lower levels of the machine in an evolutionary manner, providing a consistently competitive system.

System Processor Technology

Describes the advances in chip and circuit design used in the AS/400 System Processor design.

Delbert R. Cecchi and Robert F. Lembach

Introduction

The AS/400™ System Processors incorporate the most advanced technologies available. These technologies allowed reducing the processor to a single card containing the processor logic, the writable control storage, and the virtual address translation mechanism.

The use of these, the densest gate array and standard cell chips ever used in an IBM processor, required advances in circuit design, logic design and verification tools, and physical design tools. These advanced tools provided the means to design, verify, test, and manage the processor design in an organized and productive manner within the IBM Engineering Design System[1].

The System Processor design features an innovative dual implementation. The same basic design was implemented in bipolar gate arrays having over 14,000 equivalent gates per chip, and a 1.0 micron CMOS standard cell family having up to 40,000 equivalent gates per chip. (Figure 1 shows a 14,000-gate bipolar logic chip, while Figure 2 shows a 40,000-gate CMOS logic chip.) These advances have allowed the entire bipolar 9406 System Processor, consisting of 86,000 equivalent circuits, to be packaged on a single card, including the high-speed static random access memory (RAM) for the control storage and the look-aside buffer. The CMOS implementation (9404 System Processor) adds one input/output (I/O) bus and the base 4 megabytes of main storage on the same card as the System Processor, bringing the circuit count on the card to 150,000.

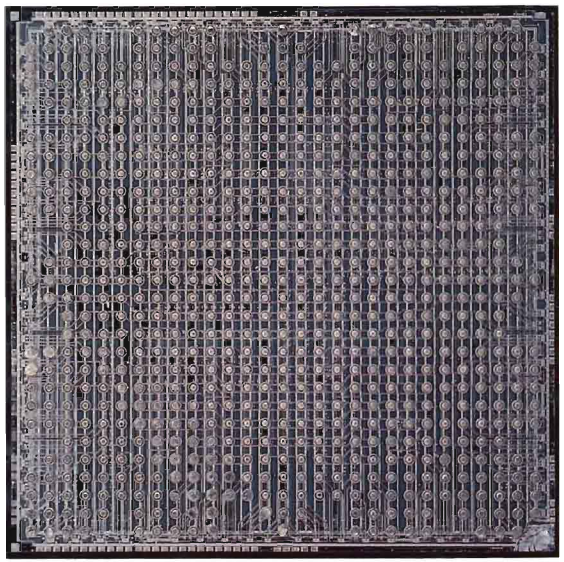


Figure 1 14,000-Gate Bipolar Logic Chip

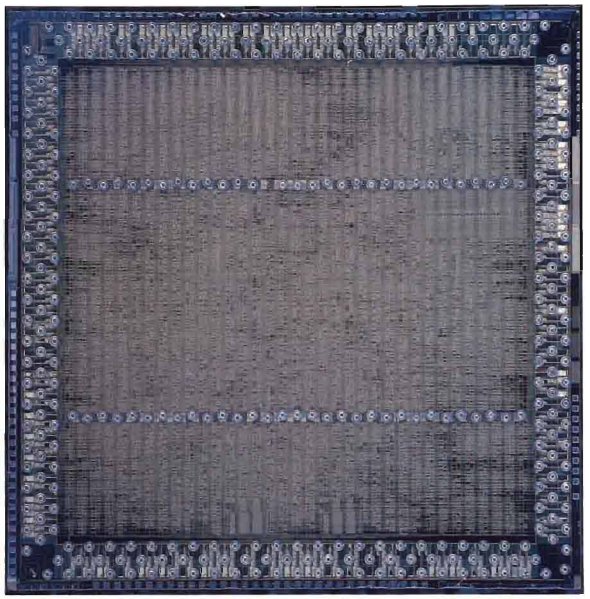


Figure 2 40,000-Gate CMOS Logic Chip

Chip Logic Technologies

The bipolar circuit technology used in the logic chips in the larger models of the AS/400 system is an enhanced version of a family of technologies used on the IBM System/36 5360 Model D processor [2]. It uses a transistor-transistor logic (TTL) circuit family in a 2.5 micron oxide-isolated bipolar process with four levels of metal. Each internal cell on the chip contains five transistors and five resistors. This allows the construction of a four-way NAND in one of four power levels or 1 bit of RAM in a single cell. To make the best use of available components and increase the density and performance of designs on the bipolar gate arrays, a number of special-purpose macros were designed, including RAMs, a general-purpose register stack, and a 9-bit parity checker and generator. These special macros were jointly developed by IBM, East Fishkill, NY, and IBM, Rochester, MN.

While the bipolar implementation was able to achieve a 60-nanosecond cycle time and, therefore, the performance necessary for the larger models of the AS/400 system, a less expensive and easier-to-cool logic chip implementation was desirable for the smaller models, while preserving the investment in software and microcode. Due to its dramatically lower power requirements, single-supply operation, higher density, and lower per-circuit

cost, a 1.0-micron CMOS process developed by IBM, Burlington, VT, was chosen as the technology for the smaller models. A standard cell design system was jointly developed by IBM Burlington and IBM Rochester [3]. Thus, it was possible to convert the bipolar logic design to CMOS.

In both technologies, I/O circuits were designed with controlled transition drivers and high noise-tolerant receivers to maximize the number of drivers that can be switched at the same time and minimize any chance of noise causing an error. Both technologies also allow the construction of storage arrays on the chip. The logic chips are packaged in single-chip modules, having up to 264 pins on 2.54mm centers. Heat sinks and internal thermal enhancements are used on the higher-powered chips to minimize the chips' junction temperature and maximize reliability. The characteristics of both technologies are shown in Figure 3.

While many of the interfaces used in the system are TTL-compatible, such as that to the main storage cards, extensive analysis revealed the benefits of using a smaller swing where not prohibited by compatibility concerns. A joint effort with the technology developers in IBM Burlington and IBM East Fishkill resulted in the definition of a new set of signal levels. Though similar to TTL, the up level from the driver is controlled carefully to be between 1.55 and 2.2 volts, while the down level remains at 0.5 volts. The maximum signal transition is thus 1.7 volts, or about half of a typical TTL transition. It is therefore possible to drive a full transition over a normal 80-ohm printed-circuit transmission line with a 16 mA driver. The transient current in the power distribution network is also halved compared to TTL drivers having the same performance.

Chip Array Technologies

The large arrays, such as the horizontal control storage arrays, are implemented in stand-alone

Logic	Bipolar	CMOS
Technology	2.5μ4LM	1.0μn-well DLM
Cells (wirable)	7250 Internal	27,720
Circuits (equiv. 2w)	14,000 (max)	40,000 (max)
Delay/Ckt (W C)	1.4 ns	2.1 ns
Power/Ckt	.54 mw	.1 mw
Power/Chip	7 watts	1.5 watts
Imbedded Arrays	Personalized	Custom
Metal Pitch	6.6 microns	3.3 microns
Size	7.4 mm sq.	9.4 mm sq.
I/O	240	231
RAM		
Chip Bits	18K	144K
Module Bits	72K	288K
Access Time	20 ns	30 ns

RSLL396-3

Figure 3 Comparison of Logic and Array Technologies

RAM chips. A companion set of static RAM modules was provided by IBM Burlington. The bipolar RAM chip is organized 2Kx10, with 9 bits being used. It is packaged with up to four chips in a 24mm-pin grid-array module. The chip is fabricated in a 2-micron trench-isolated process and dissipates less than 1 watt in standby (worst case) and 1.4 watts when active.

The CMOS static RAM is implemented in nearly the same 1.0-micron CMOS process used for the logic chip. It has an access time of 30 nanoseconds and is available in one- or two-chip modules. The characteristics of both these RAM technologies are shown in Figure 3.

Methodology

Once the CMOS design system and process capability was in place, and the bipolar design was reasonably stable, it was necessary to convert the logic description from the bipolar book set to an equivalent CMOS one. (The individual types of gates available in a design system are known as **books**, which together form the library for a given technology.) Of necessity, the book sets are different. A set of programs, developed using the Logic Transformation System (LTS), is able to convert a design from one circuit family to another automatically, while taking into account the differences in function, fan-in, fan-out, and so forth. Different algorithms are available to optimize the resulting design, depending on the needs of the designer. The new design is compared to the old design; conceptually, the truth tables (or boolean equations) for both designs are derived and compared for equivalence. The simulation test cases are re-run, and the timing is optimized to correct any deficiencies. (For more information, see the article *VLSI Design Process for the System Processor*.)

The chips can be fully tested using Level-Sensitive Scan Design (LSSD). LSSD is a design technique for enhancing the ability to test logic chips by connecting all of the latches on the chip serially into one or more shift registers, or scan rings. Test patterns are then shifted into, and results shifted out of, the scan rings through special test lines. This makes all internal-state information available and controllable during the test. As a result, test coverage is better than 99.5%. While scanning the patterns, a special test input is activated that holds the drivers in a high-impedence state, to eliminate the noise caused by large numbers of drivers switching at high frequency during the scan process. Special circuitry is also incorporated on the chips to make the RAM macros accessible directly from the pins for AC testing.

To use the CMOS technology, it was necessary to provide equivalent macros to those available in the bipolar technology. Rather than design each of the RAM macros individually, macros were compiled to meet the designer's size and performance specifications. These RAM macros are completely compatible with the rest of the design system elements. Space is left in the macros for wires to pass through to reduce wiring congestion and allow placement flexibility. The system automatically generates all of the design system rules so a designer can use RAM macros like any other library element.

Optimizing the Logic Chip Physical Design

Physical design has traditionally consumed 10 to 20% of the computer time used while designing a chip. Traditional chip physical design practices were strained in an environment using both bipolar and CMOS circuit families, with multiple chip images per family. To keep pace with the growth in the number of circuits on a chip, placement and wiring algorithms were required that scaled well computationally and produced consistently high-quality results. The goal of increasing process automation and efficient algorithm use demanded dependable physical design methodologies.

Circuit placement directly affects performance and density, and so was one key process. Wire length, wiring congestion, critical nets, and pin-density metrics were optimized during placement. Constraints included preplacement biases, widely varying sizes of circuits and macros, simultaneous switch, clock skew, and second-level package wirability. Placement of circuits on these gate-array and standard-cell chips is a hard combinatorial optimization problem due to these many, often conflicting, metrics used to judge solution quality.

A unified circuit-placement approach was used to manage this complexity across the different chip images and circuit families. This unity was based on simulated annealing [4], a multivariate optimization algorithm developed within IBM and integrated into IBM's Engineering Design System. In Rochester, the advantage of this novel algorithmic approach in a production environment was seen as early as 1983.

Optimization algorithms are judged by their relative time complexity, such as their speed and ability to scale with problem size, and by their relative performance, such as the level of solution quality. Simulated annealing was found to scale as N, where N is the number of circuits to be placed. For the bipolar and CMOS chips, typically 10- to 20-million moves were made on the 5,000 to 10,000 various-sized circuits per chip during placement evolution. Chip wire-lengths fell 10 to 25% compared to prior constructive and iterative algorithms in similar run times. This lower wire demand required less total time for the wiring task, yielded a 50% reduction in wiring overflows, and resulted in fewer timing problems. Overall physical design cycle time was reduced, on average, by 25% compared to prior system designs.

The control of timing critical paths, the placement of both large and small objects, the capability to perform quick incremental changes, and the ability to restrict circuits to specific areas to bound simultaneous switching and enhance card wirability placed additional burdens on the physical design methodology.

Control of critical logic timing, such as clock trees, was accomplished with minimum and maximum capacitance goals applied during placement evolution. On the bipolar chips, capacitance limits existed on all nets due to timing goals and technology restrictions on the maximum net capacitance each circuit could tolerate. With this approach, all nets were viewed as critical, with the priority being application-specific.

The presence of both large and small objects complicated the chip physical design task. This task was not partitioned into subtasks because the interaction of the circuits was, in general, not disjoint. As such, placement was allowed to move all objects under the guidelines of suitable metrics, such as macro blockages. Experiments showed that the resulting solutions mimic manually generated solutions, and often yielded novel solutions. After placement, chip plots were made with the circuits colored based on their function to provide insight into the underlying hierarchical logic structure and to yield evidence of reasonable chip floor plans. Because placement is a dynamic activity, a videotape of several thousand circuits being placed was produced to better observe and understand placement evolution for both random-logic and macro-dominated designs.

The capacity to manage incremental physical design changes is important due to the parallel design activities in which chip physical design coincides with system simulation. The incremental change strategy attempted to minimize disruption of existing placement and wiring for circuits that were already timed. Incremental placement inserted new circuits into available positions while being guided by a minimum wire length goal. Incremental wiring used mazerouting and manual embedding.

In some cases, the physical design influenced the logic design. As the logic evolves from the hardware description language though synthesis and ends up in a target technology, logically equivalent signal sources are connected to logically equivalent signal destinations in an arbitrary fashion that may aid or hinder physical design. Reordering these equivalencies was performed after circuit placement to reduce wire length. Scan paths were modeled as travelling salesman problems and solved using simulated annealing. For the case of equivalent sources driving equivalent sinks, such as in repowering

trees, a simulated annealing program was developed to reassign these connections based on minimum wire length and balanced loading. Typically, 150 unique reordering sets were manipulated. Total chip wire length was reduced up to 10% using both of these programs.

I/O circuits were grouped into specific areas for several reasons. Simultaneous switching was an important consideration due to the large number of chip I/O, and the potentially unpredictable results from large current pulses causing noise coupling to nearby quiet drivers or receivers during the switching time. By controlling the placement of the buses, the electrical noise generated is reduced considerably, enhancing the reliability. Intelligent grouping of signals also enhanced card wirability.

Conclusions

The design of the AS/400 System Processors using state-of-the-art technologies required many advances. The semiconductor process development, the physical design system, and the system logic design were done in parallel, requiring close interaction between technology developers and system designers. For example, while the semiconductor manufacturing process was still under development, hundreds of circuit books were designed and simulated. At the same time, major processor implementation decisions, such as the partitioning of the function into individual chips, setting the clock cycle time, and designing the packaging, were made. Because modification of dense chips to correct mistakes is impossible without another chip pass, great emphasis was placed on the accuracy of the design in all aspects. Successfully running the operating system on the first-pass design, at the designated cycle time, was proof of our methodology.

Acknowledgments

Significant contributions to the technology and tools were made by individuals at several IBM locations. Their efforts are truly appreciated.

References

1. Freeman, W.J. III and V.J.Freund, Jr., *A History of Semicustom Design at IBM*, **VLSI Systems Design's Semicustom Design Guide,** 1986.
2. Brenner, S. et al, *A 10,000 Gate Bipolar VLSI Masterslice Utilizing Four Levels of Metal,* **1983 ISSCC Digest of Technical Papers.**
3. Aldridge, A. et al, *A 40K Equivalent Gate CMOS Standard Cell,* Custom Integrated Circuits Conference, Portland OR. 1987.
4. Kirkpatrick, S., C.D. Gelatt, Jr., and M.P. Vecchi, *Optimization by Simulated Annealing*, **Science,** Volume 220, Number 4598. May 13, 1983.

VLSI Design Process for the System Processor

Describes the unique methodology by which the System Processor was designed and verified.

James R. Rubish, Larry F. Saunders, Timothy J. Mullins, and William J. Goetzinger

Introduction

In today's world of advanced information processing and very large scale integration (VLSI) logic designs, automated methods of design and verification are essential. Many motivating factors drive one overall design goal: to obtain functional hardware on the first attempt. One of these motivating factors involves reducing the time and cost of obtaining the high-technology VLSI prototypes required for laboratory testing. To obtain this functional first-pass hardware, high levels of quality must be achieved in logic entry, timing evaluation, and functional verification.

The VLSI design process used for the AS/400™ System Processor is shown in Figure 1. Once the definition for the specific processor architecture has been completed, the next task is to translate the ideas, timing diagrams, and data flow pictures from the functional specifications into an actual logic design (AND and OR gates). This logic entry must be done in a timely manner with a high degree of quality to ensure the integrity of the original definition. The methodology for logic entry in the AS/400 System Processor involves both high-level and low-level descriptions.

Once the logic entry phase is completed, verifying that resultant logic is the next task. This process involves the use of timing evaluation and functional verification. This may be the most critical phase, in that a one-pass design is improbable without large amounts of testing before the hardware is actually built.

As verification is completed, any changes or modifications to the design are reapplied in the logic entry phase. Although this process was used in the past, significant enhancements in each of the key indicated activities leads to a higher quality design. Not only is each of the activities important by itself, but the manner by which the transfer is made from one activity to the next makes the design methodology unique for the System Processor. Once the VLSI prototype hardware is built, this design methodology is evaluated based on the resulting hardware.

Automation in each of the areas of logic entry, timing evaluation, and functional verification is a key factor in the attempt to design a functional first-pass processor exhibiting high standards of quality. Although the processor is but a small part of the system hardware, achieving quality for the entire system involves quality on each of the sublevel components. (For examples of how these quality standards are defined and met for the AS/400 system and the associated subsystems, see the article *Improved Methodology for Hardware Quality and Reliability.)*

Logic Entry

The first step in the design process of any processor is to create the definition (functional specification) of the computer hardware to be built. This definition includes information about all aspects of the computer's operation and detailed design.

When the functional specification for the AS/400 System Processor was completed, the next task was to enter this description into a functional format. As shown in Figure 1, a high-level description and a low-level description were created. Each of these formats uses a different hardware-description language (HDL) to illustrate itself. HDL's are specially adapted computer languages used to describe the workings of digital computers. AS/400 logic models, to be described with these HDL's and stored in an IBM Engineering Design System (EDS) computer data base [1], form the basis for all AS/400 processor designs, verification, and manufacturing. All future work with the processor design is established with a variety of different IBM EDS computer programs, all of which operate on one of the two logic models created. This work includes sublevel simulation, timing analysis, design-rule checking, test pattern generation, and other logic verifications. By completing this work in a model environment on a computer, the need to actually build hardware to ensure the processor's proper operation is eliminated. This type of automation not only saves time, but greatly reduces the possibility of human error, thus enhancing the quality of the final product.

The logic models forming the System Processor were written in two different languages: system design language (SDL) and basic design language for structure (BDL/S) [2]. SDL, the high-level language, was used to describe abstract logic behavior while omitting the underlying design details. BDL/S, the lower-level language, was a netlist description used to describe logic structures on an integrated circuit chip for one given technology.

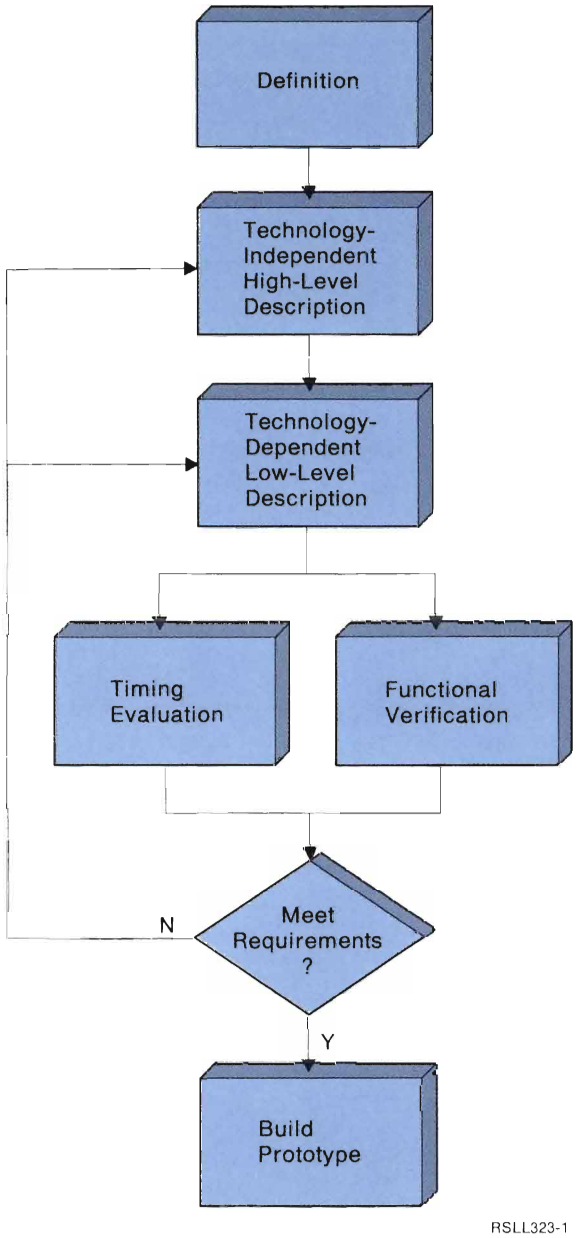


Figure 1 AS/400 VLSI Design Process

The strategy used for HDLs on the System Processors is analogous to how languages are used in computer programming. SDL is similar to a programming language such as FORTRAN or PL/I. These languages are independent of any particular computer, and can be used on any computer having an appropriate language compiler. The language compiler translates the programming language statements into machine-specific code. SDL, in a similar way, describes logic behavior in an abstract manner independent of a specific technology, but can be translated into any one of several technology-specific logic structures. High-level languages such as FORTRAN and SDL are generally easier and faster to write than low-level languages, and are conceptually easier to understand. BDL/S is similar to assembler language. Assembler language is dependent on a particular computer, and may only run on that computer. BDL/S, in the same fashion, describes the logic structure of a specific integrated chip technology.

Because large design changes in SDL are very easy to accommodate, functional verification of the SDL is desirable before the conversion is made from SDL to BDL/S. This highly detailed sublevel simulation is accomplished using the IBM EDS variable mesh simulator [3]. This simulation involves writing a preliminary set of test cases, each exercising a different aspect of the processor's operation, and applying them to the SDL model using the variable mesh simulator. Upon successful completion of this sublevel simulation, the SDL logic model can be translated into the low-level BDL/S model. This translation can be accomplished either automatically using computer translations, or using a manual process involving a language rewrite by the computer designers.

The automatic process of converting SDL to BDL/S is known as logic synthesis [4]. This is accomplished using many levels of computer programs to convert the SDL behavioral description into a technology-specific logic structure. Using this process, the technology-independent SDL language can automatically be converted to the technology-specific BDL/S language. This process is very fast, with a total VLSI design being converted in only minutes. This is analogous to using a FORTRAN compiler to convert the high-level computer-independent FORTRAN language into low-level computer-dependent assembler language. Advantages of using synthesis to convert to technology-specific BDL/S include the speed at which this can be performed, and the ability to write one HDL model and then synthesize to several different target technologies. The AS/400 system, using more than one VLSI technology over its range of processor models, provides an example of how a different VLSI technology can be used with no redesign needed.

A second method of converting SDL to BDL/S is through the manual translation process. In this method, the designer identifies the underlying technology-specific logic structure implied by the SDL, and then writes out the equivalent BDL/S statements by hand. This is used when the specific implementation desired by the designer is not equivalent to the implementation received through the automatic translation tools. This may result when the method of implementation chosen by the synthesis programmer is different from the method of implementation chosen by the design engineer. One advantage of using manual translation is the ability to match the functional needs of the design to a precise physical characteristic of a technology. This may include a physical size or functional speed characteristic of that technology. Both synthesis and manual translations were used to design the AS/400 System Processor.

Timing Evaluation

Upon completion of the logic entry phase, the resultant BDL/S must be verified considering the machine cycle time. As shown in Figure 1, timing evaluation becomes one of the next phases in the VLSI design process. For the AS/400 system, close timing tolerances were used to maximize

the system performance while minimizing the required hardware. Special logic circuits have been developed outside the standard technology set, and the clocking scheme has been tailored to favor longer logic paths in the design. This was accomplished by building in a small amount of overlap into adjacent clocks that define a machine cycle boundary. Extensive machine timing analysis (TA) becomes a critical aspect in achieving a one-pass functional design. By the conclusion of the development process, all System Processor timing requirements were satisfied to statistical worst-case limits, a significant accomplishment in view of the cost/performance approach to the design.

AS/400 delay analysis was based on the IBM Engineering Design System timing analysis (EDSTA) and the IBM early timing estimator (ETE) program sets. Statistical values and internal circuit delay equations implemented in the TA delay rules are provided by a circuit development group. Off-chip driver delays are calculated using a circuit-model analysis program which takes into account the details of the net configuration on the circuit card. The analysis programs are flexible enough to allow timing refinements at various stages of the design. Such stages include pre-wired chips, circuits placed within a chip, circuits completely wired in the chip, and chips wired on a circuit card.

The scope of the AS/400 model evaluated by TA encompassed the processor complex that packages the logic chips, the control storage arrays, and all main storage cards. All interfaces among these units were evaluated for delay characteristics. The boundary of the design is the asynchronous input/output (I/O) bus interface. Two types of TA runs evaluate the logic timings of the processor design. Figure 2 illustrates the basic clocking scheme and timing requirements. Late Mode TA checks for logic delays that exceed the latch-to-latch timing limit imposed by the machine

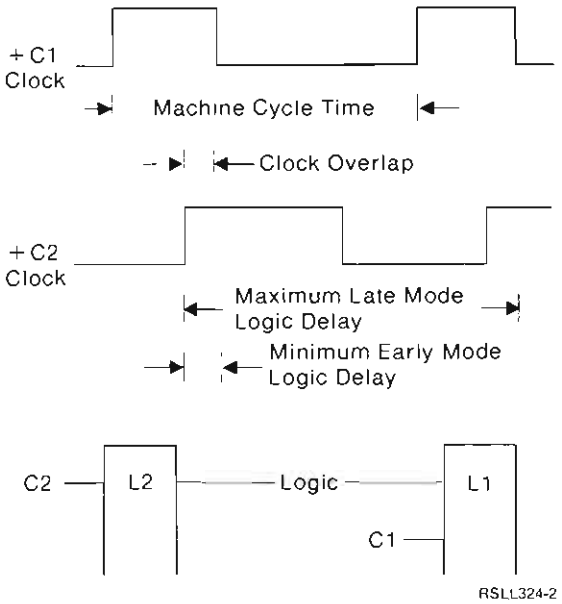


Figure 2 Basic AS/400 Clocking Scheme

cycle time. Such a timing path is initiated by a C2 clock and captured by a C1 clock, and is the Maximum Late Mode Logic Delay identified in Figure 2. Early Mode TA checks for logic that does not meet the minimum delay required between two latches, which is also identified in Figure 2. Through the use of TA, the final AS/400 processor design met all timing requirements. Correct Late Mode and Early Mode function was guaranteed.

The AS/400 System Processor uses several bidirectional buses that were implemented using three-state devices rather than open-collector circuits. Although this achieves a performance gain on the affected busses, a more in-depth analysis must be completed when using these devices. Logic simulation does evaluate some aspects of bus contention when using a three-state device; however, this analysis does not check within the scope of a single machine cycle. For the System Processor, TA ensured that no driver-overlap problems existed where three-state control mechanisms were implemented.

One such bus control method implemented involves the gating of a bus enable/disable decode signal with a free-running oscillator pulse. Standard timing tests at the point where decode and oscillator logically combine verify that the logic signal arrives satisfactorily with respect to the oscillator edge.

A second bus control mechanism involves direct control of driver-enable signals by latch outputs and combinational logic. This is a more complex case to model. Path delays are evaluated using switching times for both voltage threshold-based delays for off-chip drivers, as well as current threshold-based delays. Through the AS/400 TA work, fully functional, reliable, and high-performing bidirectional bus interfaces were guaranteed.

Finally, timing evaluation was used to quantify off-chip driver switching activity. Delay-analysis results determined the degree of coincidence of switching drivers. Because of the large number of off-chip drivers available on each logic chip, limits are placed on simultaneously switching drivers to avoid induced noise. An analysis program was developed to interpret the delay times at chip outputs, and then plot switching activity as a function of time. This invaluable aid allowed each chip design to achieve maximum switching activity while not exceeding limits imposed by induced noise concerns. As a result, design hazards caused by induced noise have been thoroughly investigated, and an extremely sound, reliable processor implementation was produced.

Functional Verification

In parallel with the timing evaluation phase shown in Figure 1, the logic must also be functionally verified. Although sublevel simulation has been completed, a more sophisticated means of functional verification must be used to model the entire system. The challenge undertaken by the System Processor verification group was to achieve the goal of a single-pass design given the complexity of the Processor.

To meet this challenge, a system-simulation philosophy was adopted. This concept includes surrounding the System Processor by the other system components and modeling them in a simulation environment. High-level test cases are run to verify the function of the design using the actual system microcode. This microcode is loaded into the processor model, which is the actual BDL/S model received from the logic entry phase. By running test cases in this way, a detailed and realistic system environment is produced, and any problems found can be corrected before prototypes are actually built.

Although this may sound simple, a sophisticated means of implementing this concept is required. The simulation vehicle used must possess the speed and capacity necessary to perform this type of system simulation. A review of state-of-the-art simulation methods culminated in the selection of the Engineering Verification Engine (EVE) [5] as the AS/400 simulation method. EVE is a hardware simulation engine: a specialized, highly parallel computer developed specifically for the simulation of hardware designs. The ability to simulate hundreds of thousands of gates, combined with the speed necessary to run millions of instruction cycles, made EVE an ideal choice for system simulation.

With the selection of EVE, system simulation evolved into two components. The first, internal microprogramming interface (IMPI) simulation, verifies that the AS/400 architecture is implemented correctly. The second, bus simulation, ensures that the processor properly interacts with other system components (I/O devices, for example).

IMPI simulation actually imitates the debugging work that was done later in the laboratory during initial system bringup. The processor model was loaded with the same horizontal microcode (HMC) that is used on all AS/400 systems. IMPI test cases, the same as those used by the HMC developers to verify both the microcode and the hardware, were run on the simulation model. Those extensive, high-level test cases provided the measure of whether the System Processor design adhered to the architecture specifications.

Although IMPI simulation is used to verify internal operations, the external processor interfaces must also be exercised to ensure their validity. Because of this, bus simulation was developed to ensure the System Processor I/O channels were implemented correctly. This was accomplished by coupling the System Processor model to other models for various I/O devices. To give credibility to the simulation, these I/O models were derived from actual device designs. Bus simulation was used to test initial program load (IPL), direct memory access (DMA), bus error sequences, and various functions used for hardware problem debugging. It ensured that the system could be initialized to the run state, from which normal system processing could occur. Bus simulation also verifies that various debugging facilities were operational, should they be needed during the laboratory bringup that followed.

Using IMPI simulation and bus simulation as parts of the overall system simulation strategy undoubtedly saved time and resources, as well as improved the overall quality of the System Processor.

Conclusions

The motivation to obtain functional first-pass hardware has brought about changes and success to the VLSI design process.

Logic entry has evolved so the designer is removed from the trivial details of technology-dependent issues to allow more emphasis on architectural advancement. The timing results seen at the completion of the project met all the requirements set in the beginning. All logic paths were implemented successfully to achieve functional timings under worst-case conditions. The system simulation method of functional verification successfully operated the hardware, microcode, and test cases together as a system before any parts were built. This allowed the laboratory bringup to become the last step in the verification process, not just the start of hardware debugging.

The resultant hardware obtained from this process was functional on the first pass. This functional hardware was then given to the programming groups, allowing their efforts to be undertaken shortly after receipt of the first-pass VLSI parts. This design process provided significant improvements over past designs for scheduling, productivity, and above all, quality. It has set the standard by which future designs will be measured.

References

1. Dunn, L.N., *An Overview of the Design and Verification Subsystem of the Engineering Design System*, **Proceedings of the 20th Design Automation Conference,** Miami, 237-238. June, 1983.
2. Maissel, L.I. and H. Ofek, *Hardware Design and Description Languages in IBM*, **IBM Journal of Research and Development,** Volume 28, Number 5, 557-563. September, 1984.
3. Case, P.W. et al, *Design Automation in IBM*, **IBM Journal of Research and Development,** Volume 25, Number 5, 631-646. September, 1981.
4. Saunders, L.F., *An Approach to VLSI Logic Design*, **Proceedings of European Conference on Electronic Design Automation (EDA '84),** Conference Publication 232, 33-34. March, 1984.
5. Blank, T., *A Survey of Hardware Accelerators Used in Computer-Aided Design*, **IEEE Design and Test of Computers,** Volume 1, Number 3, 21-39. August, 1984.

Performance Analysis of the System Processor

Describes the techniques used, primarily statistical modeling methods, to ensure that AS/400 performance requirements were met.

Harold F. Kossman and Merle E. Houdek

Introduction

Applications are becoming more complex, increasing the path length run to perform a given function. The use of application generators usually creates less-efficient code and further increases the path length. The development of user-oriented systems is also required for user productivity. The net result is a requirement for a high-performance processor.

One of the integral parts of processor development activity is processor performance analysis. This analysis started when the processor data flow concepts were generated, and continued through the detailed design and build phases of processor development. The resulting AS/400™ System Processor design not only met the processor performance objectives, but achieved the best performance for the technology used for the processor design. This processor allowed a user-oriented system to be built with the capability to run more complex applications.

Methodology

A frequently used method for simulation-performance modeling consists of using an existing machine to generate a trace of instructions and main storage accesses, and then writing a program simulating the hardware to process those instructions and use those main storage accesses. This method is fine when the instruction set the machine processes is small or when the time available to do the analysis is great. On a processor that uses horizontal microcode (HMC) to process the instructions, all microcode must be available for all instructions before the instruction trace can be run. If the processor's instruction set is small or the processor architecture does not require significant change to the microcode, this does not present a serious problem. However, the AS/400 internal microprogramming interface (IMPI) instruction set contains over 250 instructions and the AS/400 architecture changed extensively to satisfy required performance objectives. (See the article *System Processor Architecture* for more information.) With an instruction set that large, microcode development for all instructions is a very large task and is typically not complete until late in the development cycle. It is not possible to generate that amount of microcode quickly to evaluate various processor architectural alternatives. Therefore, two models were created: one very detailed simulation model used for microcode development and verification, and another simulation model that used a statistical approach to generate instructions and main storage accesses. Though both models provide input to the development process, the statistical model provides needed input early in the process (see Figure 1).

To optimize the time to develop and debug the model, the model was written in A Programming Language (APL). APL allows for efficient manipulation of matrixes, which is desirable when using a queuing structured model. The model was run on an IBM 3081 Model D, with 32 megabytes of storage; the model used 35% of the processor for approximately 45 minutes. Again, to decrease debugging time and increase the time to make

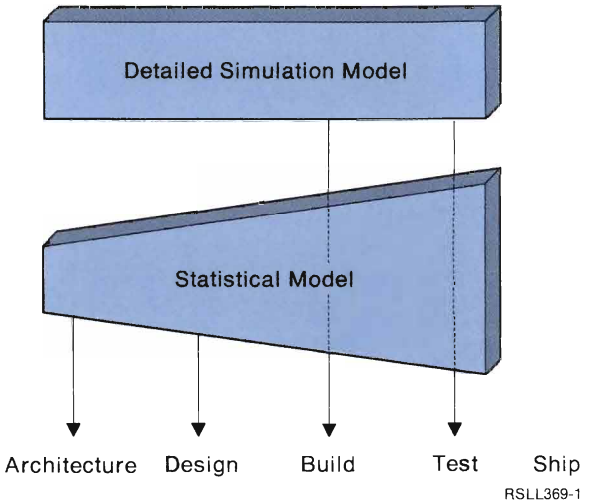


Figure 1 Impact of Two Modeling Methods on the Product Cycle

corrections and alterations for different conditions, the run time was considered a necessary expense.

The statistical method consisted of analyzing an instruction mix and a main storage trace and extracting the important characteristics affecting performance from each. Then, the frequency in which these characteristics would occur was predicted. A model was created that used these frequencies or statistics, replacing the need to analyze actual traces. The model was then run against these statistics, generating an average instruction time. The statistics were individually changed to determine their sensitivity. When one was found to be unacceptably sensitive, it was replaced with a more detailed, modeled description of that facility.

Inputs to the model included instruction mixes, HMC for the instructions, statistics concerning locality of reference of main storage (for both data and instructions), and hardware structures and timings (see Figure 2).

Instruction Mix

The instruction mixes and main storage trace statistics were determined empirically by running a large set of applications on an IBM System/38 Model 700 and collecting instruction usage and main storage usage statistics. Choosing benchmarks to run while collecting statistics is a difficult decision. Many different benchmarks were used, including actual customer applications, as well as synthetic benchmarks developed internally, in an attempt to establish a representative field of applications that tested all aspects of the system. Five applications were selected and run, and instruction-usage data was collected for each application. (A total of 2 billion instructions was accumulated from the five applications.) The individual application data was then normalized and combined, such that the resulting mix was a single list of instructions, weighted and ordered by contribution, to the resulting average instruction time. This new instruction mix was used to make the overall prediction for average instruction time. However, individual application mixes were also used for studies of specific areas.

When analyzing these instruction traces, the top 10 instructions (figured by contribution to the average instruction time) contributed over 40% of the total average instruction time. The top 25 instructions contributed 60% and the top 40 instructions contributed over 70% of the total average instruction time. To analyze the various architectural alternatives, a significant confidence factor could be obtained by running a subset of the total 250 instructions in a mix based upon the probability of occurrence of each instruction. In addition, many instructions had multiple path

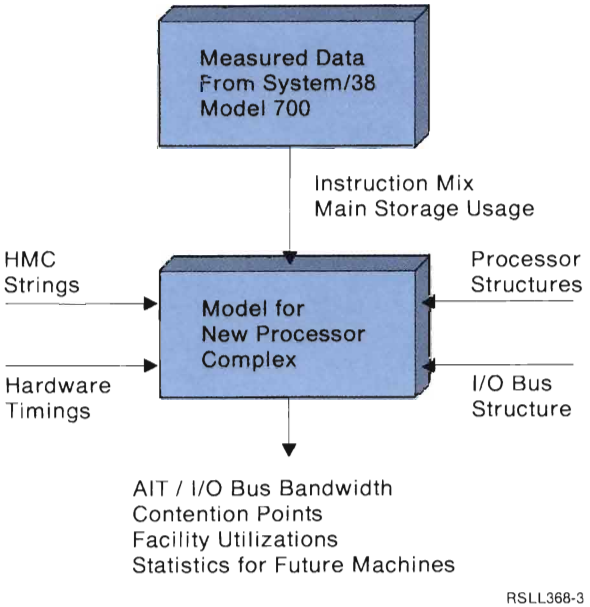


Figure 2 Information Inputs for Processor Performance Analysis

lengths, such as the move-character instruction. The various paths were statistically analyzed for frequency of occurrences and for the contribution to the overall average instruction time to determine if they needed to be broken down further into several representative paths. The number of instructions run in the model varied as the degree of confidence in the design became more secure. Early in the cycle, when high-level architectural decisions were being made, few instructions would be used; as the confidence increased, additional instructions were added as the microcode became available.

Model Generation

When the instruction mix was determined, the engineers responsible for the HMC started writing the microcode for the instructions that contributed the most to the total average instruction time. Because the primary focus of the analysis was to generate a preliminary analysis of the performance for the proposed processor architecture, pseudo-microcode was developed. This pseudo-microcode allowed a level of simulation that ignored non-performance details, but still captured all pertinent information necessary to analyze System Processor performance. And, because this pseudo-microcode was a subset of the actual microcode, it was much easier to change. Therefore, alternative microcode strings could be quickly generated and tested. Initially, the few instructions that provided the greatest contribution to the average instruction time were coded and were able to provide a statistically significant confidence factor in the accuracy of the results. As the proposal for the processor architecture was finalized, and the microcode became available for additional instructions, these instructions were added to the model to improve the average instruction time contribution and confidence.

As this progressed, the capabilities of the proposed hardware facilities and timings were being modeled. The System Processor, the virtual address translator, and the main storage unit run asynchronously with respect to each other. The virtual address translator operations and the main storage accesses can overlap with the control-word processing within the System Processor, and the degree of overlap depends upon the microcode sequences. In addition, within the main storage unit, different main storage accesses also operate somewhat independently with respect to each other. The average instruction time model simulated the processing time of microcode sequences and all of the interactions between the System Processor, the virtual address translator, and main storage. Because many proposals and alternatives were expected to be evaluated, results needed to be available quickly to be most useful for design decisions. Therefore, a modular queueing structured model was chosen to

generate results quickly. This modular structure consisted of separate facilities to evaluate each independent function within the System Processor, virtual address translator, or main storage area, with easily changed parameters for scheduling processing durations for those facilities. As a facility was processing, it became unavailable for other use, though it could call other facilities. The called facility, if not busy, would then run. If the called facility was busy, the processing request and the required parameters were placed in a queue for that facility. The calling facility continued, if possible; if not, it would be placed in a hold-off situation, just as a real processor would. When a facility completed processing, it would reset its busy signal and requests on its queue would start processing.

Processor Evaluation

When the model was generated and the inputs were agreed upon, the results were evaluated. This was done in several different ways. First, traces of the modeled-processor operation were generated for the instructions run on a cycle-by-cycle basis, and the development engineers reviewed the traces against their expectations of the operation, looking for accuracy of both microcode and hardware facilities. In addition, statistical collection facilities were placed within the model to measure various parameters and utilizations. These results were then compared against previous models and differences were investigated to determine whether the difference was expected and explainable, or if the difference represented a problem in the model. When the final hardware was actually measured, the results were found to be within 1% of that predicted by the performance model, due to the intensity of the model verification process.

When the model was written, debugged, and accurate, results were generated. (For the actual methodologies, see Figure 3.) The processor performance analysis found many unexpected

Code horizontal microcode for top instructions based upon contribution to total AIT.

Generate instruction stream to be analyzed based upon measured instruction mix.

Run model processing instruction stream.

Calculate average processing time for each instruction modeled:

$$AIT_i = \sum_{j}^{\text{all paths}} (T_j / N_j) * F_j$$

T_j–Represents accumulated time for the jth path through for the ith instruction.
N_j–Represents number of times the jth path was run for the ith instruction.
F_j–Represents the frequency of occurrence for this path for the ith instruction.

Calculate contribution to AIT due to simulated instructions:

$$CONT_{sim} = \sum_{i}^{\text{number sim}} AIT_i * FREQ_i$$

$FREQ_i$–Represents frequency of occurrence of the ith instruction.

Calculate AIT ratio to S/38 Model 700 for simulated instructions:

$$AITR_{sim} = CONT_{sim} \text{ (NEW MACHINE)} / CONT_{meas} \text{ (S/38 Model 700)}$$

Assume AIT ratio for non-simulated instructions; previous experience has shown that, because design optimization occurs on the top instructions, a good approximation for bottom ops is:

$$AITR_{non\text{-}sim} = .9 * AITR_{sim}$$

Calculate contribution of non-simulated instructions:

$$CONT_{non\text{-}sim} = CONT_{meas} \text{ (S/38 Model 700)} * AITR_{non\text{-}sim}$$

Calculate total AIT:

$$AIT_{total} = CONT_{sim} \div CONT_{non\text{-}sim}$$

Calculate instruction throughput:

$$TP = 1 / AIT_{total}$$

RSLL450-1

Figure 3 Methodology for Calculating the AIT

bottlenecks and contention points that were eliminated or minimized throughout all areas of the System Processor, the virtual address translator, main storage, and the input/output (I/O) bus areas. In addition, hundreds of questions were answered for the developers concerning complexity-versus-performance tradeoffs, such as considering the performance gained if the virtual address translator process was reduced one cycle under certain conditions.

I/O Bus Evaluation

After the processor was analyzed, the processor average instruction time was analyzed after considering the I/O bus contention for main storage. In addition, the I/O bus performance was analyzed after considering main storage contention due to the System Processor. Also, because some of the system's models have multiple busses, contention for bus facilities exist as well. These questions were answered when a model of the I/O bus structure was added to the processor model, because the System Processor already included the main storage facility. A statistical simulation model of the I/O bus was developed using the same methods described for the processor model and was integrated into the processor model. The I/O bus and the System Processor were then run simultaneously several different times, with some I/O busses performing either reads or writes and other performing reads and writes. As the I/O model was run, statistics were kept of these contention points within the I/O area, and curves were generated based on the effect of contention on both the System Processor and the bus. Information from this analysis resulted in significant design changes and resulting I/O bus performance improvements.

Conclusions

A statistical model was developed to study the characteristics of the AS/400 System Processor. It used a subset of the instruction set and a statistical characterization of the main storage reference patterns. I/O requests for main storage cycles were also included to support the design of the I/O connection into the System Processor and determine its effect on Processor performance.

The design of the System Processor simulation model had many goals, including: evaluate various architectural proposals to determine which one best suited the objectives; predict performance early to indicate that the design kept pace with the objectives; provide performance evaluations of design proposals in a timely manner, to help the designers make the best design tradeoff; identify problems with the design early in the design cycle; evaluate processor I/O bus contention; and provide input for system performance analysis. This approach allowed the analysis of proposals in the architecture and design phase of the development cycle to meet these requirements.

Design of the System Service Processor

Describes the Service Processor designed specifically for initial program load (IPL) and service of the AS/400 System Processor, including the important advancements implemented in the Service Processor, such as advanced fault isolation, error reporting, and fault tolerance.

William A. Thompson and Thomas M. Walker

Introduction

The AS/400™ Service Processor provides an independent system component to start the system, including verifying and initializing the hardware, finding and loading the microcode, and starting the 9406 System Processor. Additionally, the Service Processor provides new functions, such as remote and timed power on and improved diagnostic support to the central processor for detecting, reporting, and diagnosing catastrophic failures.

The AS/400 system is designed around the system input/output (I/O) bus architecture, which connects intelligent I/O bus units to the central processor. Figure 1 provides a high-level view of the AS/400 hardware. (See also the article *The Internal Input/Output Bus*.) An I/O bus unit communicates with the System Processor and controls the devices attached to it, including magnetic media devices, work stations, and communications lines. Each I/O bus unit must be loaded with microcode to communicate with Operating System/400™ (OS/400™). All system code resides either on disk devices, which are accessed at primary initial program load (IPL), or on tape, which is accessed at alternate IPL. The I/O bus unit that controls the disk and tape devices on which the system code resides is referred to as the load-source I/O bus unit. When the system is first powered on, the Service Processor assumes control of the bus, and uses the load-source I/O bus unit to obtain its microcode, and subsequently, the System Processor microcode. The control panel is the user's first interface to power-on, power-off, IPL, and select service functions. The Service Processor provides the operating system interface to the control panel functions.

To meet reliability and serviceability requirements, each piece of the system, as it is powered on, must verify that its hardware is functioning correctly, or be verified by another system component, and must notify the Service Processor or operating system of failures. Failures that prevent a successful IPL are displayed at the control panel by the Service Processor. All other failures are logged to the system error log for later analysis by automated service functions in the operating system.

If a catastrophic failure occurs in the System Processor, it is automatically analyzed by the Service Processor and the results are displayed at the control panel.

Control Panel Interface

The control panel connects to the Service Processor (see Figure 1) and provides a simple external interface to the user for selecting the IPL source and for indicating status and error conditions. The control panel microcode interface provides the Service Processor and operating system access to control panel functions through a set of messages that enables and disables panel functions and retrieves panel and power control status. The control panel is the control point for the AS/400 power system and it directs the power system to power on and power off. The system may be powered on by the user using the power-on switch on the control panel, remotely through a modem with a special connection to the control panel, or at a user-specified time using the time-of-day clock. The Service Processor, in conjunction with the control panel, also provides the capability for automatic system restart and IPL when power is restored following an unexpected utility failure. The effects of utility failures are prevented if the system has an uninterruptible power source attached to it. Uninterruptible power source status lines can be connected to the control panel so that transition to and from the uninterruptible power source, and status of the uninterruptible power source, may be monitored and reported to the system through the Service Processor. A normal power down is initiated when an authorized user enters a power-off command at a work station. The operating system requests the Service Processor to send a power-off message to the control panel. The control panel power-off switch can also be used to power off the system; however, this is viewed as an abnormal power off and may result in extended recovery time for the next IPL.

The IPL mode is a system parameter that determines the source of the IPL code. It may be changed by the user at the control panel or by the Service Processor as directed by the system. Mode A and mode B cause a normal load from disk devices, while mode D selects a tape device as the load source. The two disk modes, A and B, provide a way for the user to manage new releases or repairs to the Service Processor or

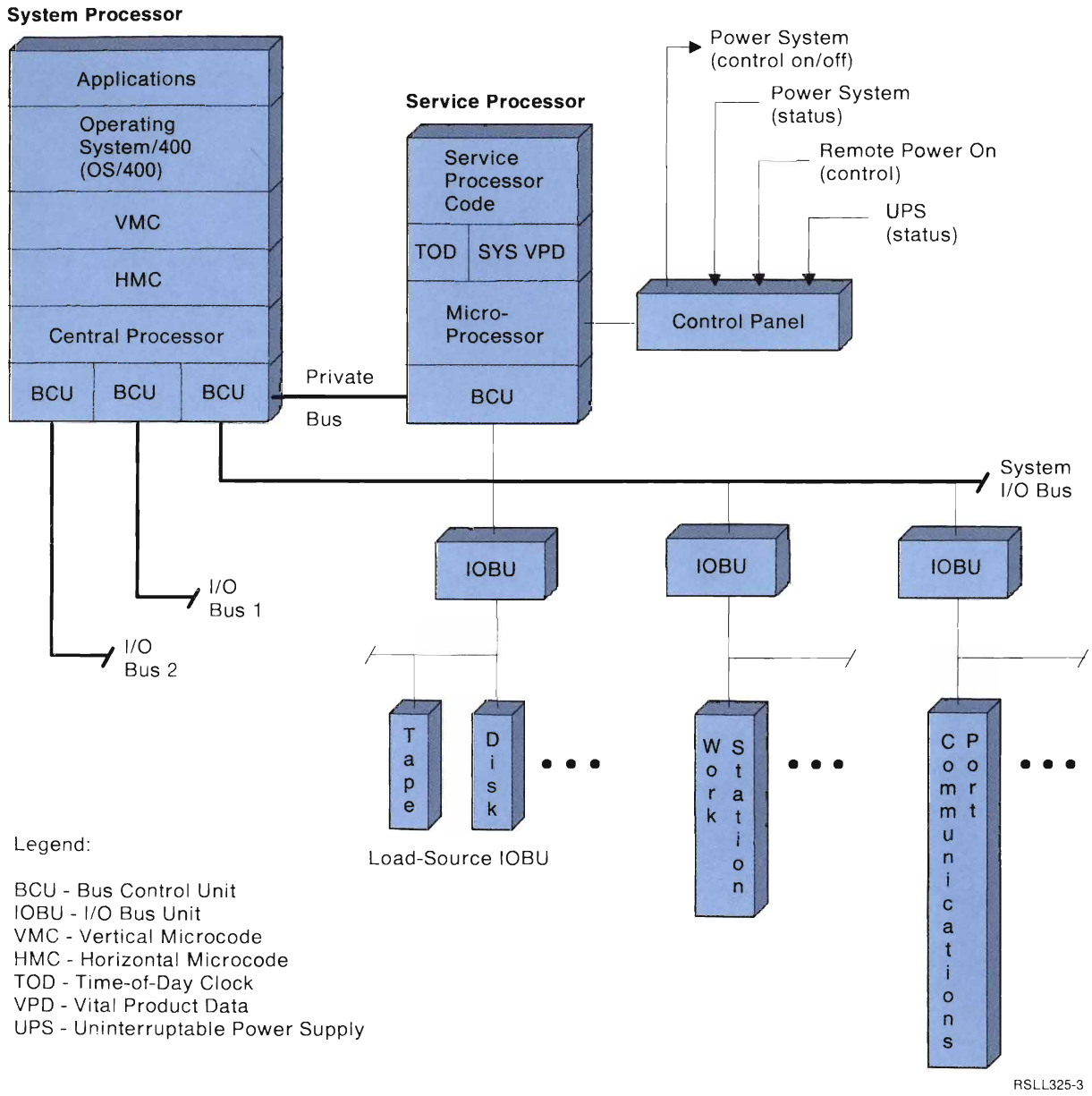


Figure 1 High-Level View of AS/400 Hardware (Model B60) and Software

System Processor microcode loads. For example, mode B can be used to test new microcode. If a problem develops, the user can perform another IPL in mode A to revert to the normal loads. This capability, to apply and back out new microcode easily without re-installing it, previously existed only in the operating system for operating system programs and application programs. Mode D is used to install or restore code to the system disk devices from the microcode saved on tape.

Bus Control

The system I/O bus is the channel connecting the Service Processor, the System Processor, and the I/O bus units (see Figure 1). One bus unit on each bus is designated bus control unit. The bus control unit provides control over arbitration, error recovery, and IPL functions on the bus. The system I/O bus has bus control unit function in the System Processor and the Service Processor, but only one is active at a time; the other functions as a standard I/O bus unit. The Service Processor assumes bus control during IPL and after catastrophic System Processor failure. Bus control is passed to the System Processor when sufficient code is loaded to provide the function.

Each I/O has some read-only storage (ROS) microcode that runs diagnostics on its hardware and I/O devices. However, to become fully operational, each I/O bus unit must load microcode into its random access memory (RAM). Bus units with disk and tape devices attached are capable of loading themselves from that storage medium. All I/O bus units must be capable of downloading their microcode from the bus control unit. All microcode loads are stored on the load-source I/O bus unit. The entire IPL process is directed by the bus control unit, which follows a command sequence across the system I/O bus to each bus unit. The Service Processor directs the loading of the load source, the Service Processor itself, and the System Processor.

Control lines between the Service Processor and System Processor provide the capability of switching the bus control from the Service Processor to the System Processor and vice versa. These control lines are referred to as the private bus (see Figure 1). The private bus is necessary because the System Processor has no ROS and cannot provide bus control function until it is loaded. The Service Processor controls the bus to access its own code and the System Processor code from the load-source I/O bus unit. The Service Processor diagnostic support uses the private bus to regain control of the bus when a catastrophic error occurs in the System Processor.

The IPL Sequence

The Service Processor contains ROS from which instructions are run as soon as the system is powered on. The ROS microcode performs the initial hardware basic assurance tests on the Service Processor. Then, this microcode performs the basic assurance tests on the system I/O bus. When this is completed, the Service Processor ROS microcode begins the search for the load-source device (disk or tape) based on the IPL mode.

For the larger models of the AS/400 system, the Service Processor must search multiple bus units on the system I/O bus. This search consists of a sequence of bus commands that first identifies the bus configuration, including location and state of each bus unit, and then queries each bus unit to find the Service Processor load. The bus unit acknowledging the query is the load-source I/O bus unit. Having located the load-source I/O bus unit, the ROS microcode now loads the Service Processor's RAM control storage with the Service Processor's run-time code and turns control over to that microcode.

For the smaller models of the AS/400 system, the Service Processor is part of the Multiple-Function I/O Processor. On these models, the Service Processor is combined with disk, diskette, tape, and communications support. This simplifies the search for the load-source I/O bus unit during IPL, because the system I/O bus need not be searched to get the Service Processor microcode load. The IPL is performed from directly attached disk or tape devices. (See the article *The Multiple-Function Input/Output Processor* for more information.)

The Service Processor RAM microcode performs some initial diagnostics on the System Processor using commands across the system I/O bus. It then directs the loading of the System Processor's control storage and main storage from the load-source I/O bus unit located earlier. The System Processor's control storage is loaded first with a sequence of basic assurance tests that verify the System Processor is functioning properly. The System Processor's vertical microcode (VMC) nucleus is loaded into main storage, followed by the run-time horizontal microcode (HMC) which is loaded into the System Processor's control storage. The System Processor is started and, when the HMC is initialized, the Service Processor microcode directs the System Processor to begin running the VMC instructions.

When VMC initialization is complete, control of the system I/O bus is transferred from the Service Processor to the System Processor's VMC. The System Processor's VMC continues the IPL sequence by loading the remaining I/O bus units on this system I/O bus and any additional I/O busses attached to the system. When catastrophic error conditions occur, the Service Processor regains control of the bus control function on the system I/O bus to run diagnostic procedures.

The Service Processor not only initializes and loads the system, it also provides for system problem analysis and reporting before the system software is loaded and when catastrophic failure prevents the system software from providing this function. During the IPL process, the Service Processor must verify that its hardware, the system I/O bus, the load-source I/O bus unit, and the System Processor are operational. It also provides specific fault information to the user in the event of a failure during this verification process.

System verification is done in a stepwise fashion, allowing each piece of the system to be verified before it is used and to report its failures to the user. This building-block method begins in the Service Processor itself, using Service Processor basic assurance tests, system reference codes, and the fault tolerance of the Service Processor.

Service Processor Basic Assurance Tests

The Service Processor basic assurance tests verify the Service Processor hardware is operational in the same stepwise fashion as the system is verified, beginning with critical hardware registers and working through all hardware interfaces. Because basic assurance tests are critical to the correct diagnoses of failures, the microcode is processed using fault-tolerant ROS to avoid Service Processor failure due to ROS failures.

In the event of a failure, the basic assurance tests isolate the failure within the failing module, and then report the error to the user in a method determined by the severity of the fault; the most severe errors are reported directly to the user through the control panel.

A special feature of Service Processor basic assurance tests is the capability to continue when hardware defects not critical to the completion of the Service Processor's main task (initializing and loading the System Processor) are detected. When these types of errors are encountered, they are logged in a software buffer and retrieved later by the System Processor's VMC.

When the Service Processor basic assurance tests have verified the Service Processor's hardware, additional Service Processor microcode continues to verify the system. In the event of a System Processor failure, diagnostics in the Service Processor determine the cause of the failure. If control of the system I/O bus has already been passed to the System Processor when the catastrophic failure occurs, bus control is retrieved using private bus control. When system I/O bus control is returned to the Service Processor, actions, such as main storage dumps and diagnostic analysis of failure data, are taken as required.

System Reference Codes

System reference codes displayed on the control panel by the Service Processor provide error information to the user. In the event of a system failure, multiple-word system reference codes are provided to diagnose the problem. They enable quick fault isolation in a user environment, and provide module fault isolation for the manufacturing environment, which reduces manufacturing costs and, in turn, helps reduce the cost to the user. The identification information available in a multiple-word system reference code can consist of the unit type, model, location, a unit reference code, device type, device model, device serial number, device location, device reference code, and other data specific to the failing unit. The Service Processor gathers this information, formats it, and displays it at the control panel. This configuration information is necessary because of the infinite number of system configurations possible.

Service Processor Fault Tolerance

The AS/400 Service Processor is a critical component of the system. The System Processor's IPL should not be halted due to minor failures of the Service Processor. For this reason, fault tolerance is designed into many critical portions of the Service Processor hardware. The Service Processor continues to function, if possible, after hardware failures are encountered.

The microprocessor gives the microcode the flexibility to use alternative methods, continue after finding an error, or choose default values in the event that proper values are unattainable from the hardware. For example, a default-value IPL mode is provided if the correct value cannot be read from the control panel. The fault-tolerant portions of the design include the RAM, vital product data, time of day, ROS, and control panel communications. For RAM, ROS, and some vital product data failures, corrective action can be taken. Failures in the time of day, control panel communications, and some parts of vital product data can be ignored. In each case, if a failure occurs, it is logged and the system IPL continues.

The Service Processor microcode attempts to recover from system I/O bus failures while it is the bus control unit. Failing bus units may be disabled by the Service Processor as needed to allow the IPL to continue. The VMC attempts to recover these units later in the IPL and may post system reference codes at the control panel or system console. The Service Processor will try several times to load the System Processor to overcome intermittent failures.

Conclusions

The Service Processor is an integral part of the AS/400 system. It provides necessary IPL functions, the interface to the control panel, and assistance with failure isolation in the System Processor. It manages many new functions including timed power on, remote power on, automatic power on after power failure, system time of day, and system vital product data.

The private bus gives the Service Processor the capabilities needed to control the system I/O bus. It can also give up that control and provide diagnostic support to the System Processor during catastrophic errors. The Service Processor operates as a bus unit independently of the System Processor, allowing needed flexibility to meet its diagnostic and IPL requirements. The Service Processor diagnostic routines provide fault isolation and display system reference codes that identify the failing unit, the error code from the unit, and the location of the failing unit within the system, resulting in faster repair times and reduced down time.

The Internal Input/Output Bus

Presents an overview of the hardware and low-level software elements of the AS/400 input/output structure.

Neil C. Berglund, John N. Tietjen, and William E. Hammer

Introduction

The input/output (I/O) structure of the AS/400™ system incorporates a new 32-bit I/O bus developed for the AS/400 system and the IBM 9370 systems. The new bus is used in both the AS/400 9404 System Unit and the AS/400 9406 System Unit. The I/O bus architecture provides communications using fixed-length messages and variable-length packet direct memory access (DMA) operations. The System Processor's hardware and software supports multiple I/O buses; the number of buses supported depends on the system model.

The AS/400 I/O bus uses an asynchronous protocol, logical addressing, and serial arbitration to provide configuration flexibility and extendibility. With few restrictions, I/O controllers can be plugged into any board socket to provide virtually unlimited configurations. For additional capacity, each bus may be serially extended to additional boards.

Emphasis was placed on providing facilities for detecting and identifying failing I/O bus units. The system can continue operation with a failing I/O controller logically removed from the configuration. Predecessor systems typically require the recurrence of a failure to locate a fault. The AS/400 I/O bus uses capture techniques to record a failure as it occurs to improve intermittent and permanent fault analysis.

I/O Hardware Structure

Figure 1 illustrates the AS/400 hardware structure. The System Processor (which includes the bus control unit), the I/O controllers, and the bus extension units are attached to the I/O bus and are called I/O bus units. I/O controllers provide disk unit, tape unit, work station, communications, and local area network I/O functions for the system.

Packaging

The 9406 System Unit is comprised of system components installed in a 1.5-meter(m) rack. Within frame structures called card enclosures, logic cards plug into zero insertion-force connectors (card slots) on horizontal boards. All card slots in a card enclosure not allocated to storage or System Processor cards are wired as standard I/O bus slots. The 9406 Models B30 and B40 provide one I/O bus with eight I/O slots in their base configurations. For increased throughput and capacity, multiple I/O buses are standard on 9406 Models B50 and B60. Model B50 has two I/O buses providing a total of 14 I/O slots in its base configuration. Model B60 has three buses providing a total of 17 I/O slots in its base configuration (see Figure 1).

Additional I/O slots are obtained with 12-slot I/O expansion card units. The expansion unit is connected to the base enclosure or another expansion unit by a cable with a card on each end. One card is plugged into the base enclosure and the other into the first slot of the expansion unit. Cable length can be up to 8 meters, permitting an I/O bus to physically span multiple racks (refer to BEU1 and BEU2 in Figure 1). Bus extension can be repeated to connect up to three expansion units to each bus supported by a processor.

The 9404 Models B10 and B20 are packaged in a versatile, low-cost system unit. The system unit, .65m (h) by .35m (w) by .75m (d), is self-contained with the System Processor, storage, I/O electronics, magnetic media devices, a power supply, and a Battery Power Unit. A single I/O bus is provided in a seven-socket logic enclosure.

I/O Bus Characteristics

The I/O bus is comprised of a 36-bit multiplexed address and data bus (32 data bits, plus 4 parity bits) and control and arbitration lines. The I/O bus operates asynchronously and uses a priority serial-arbitration mechanism. Each I/O bus can address up to 32 I/O bus units including the System Processor. Bus extension units are used to serially extend the bus and do not require a logical address like an I/O controller. Each I/O bus requires the function of a bus control unit, which is provided by the System Processor (see Figure 1). The bus control unit provides master control over arbitration, error handling, and IPL functions. I/O bus unit addresses are set by the processor software at each initial program load (IPL) to allow physical configuration flexibility. Non-volatile data in each I/O bus unit is used by the system to identify the system's I/O configuration and to provide the appropriate microcode load during IPL.

Information is exchanged between the originator of a bus operation (master) and the bus unit selected by the master (slave). The information is in the form of fixed-length messages or variable-length DMA operations. All bus units (including the processor) are capable of sending and receiving bus messages. These messages are used by the software I/O protocol to initiate and signal

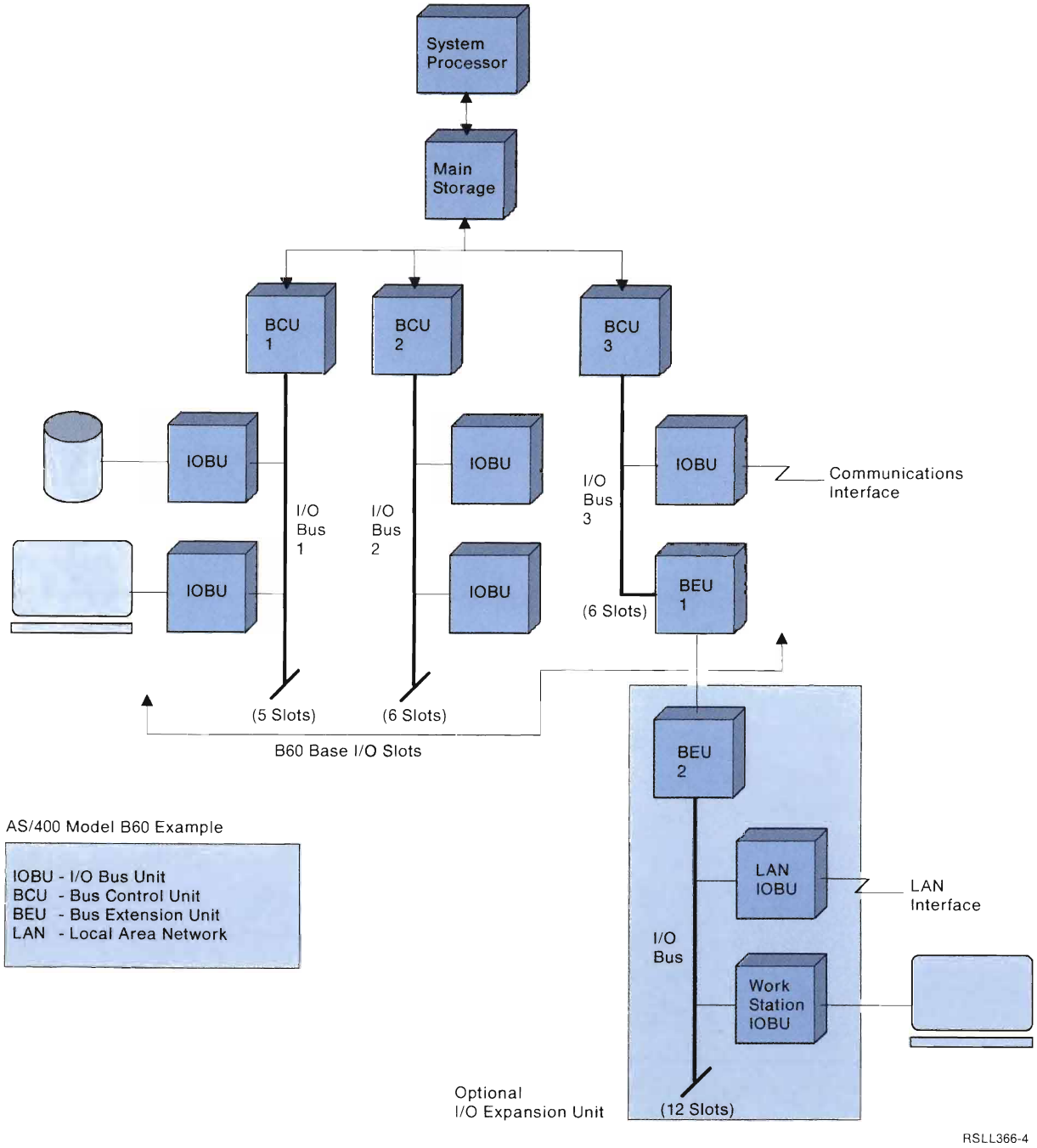


Figure 1 Sample AS/400 Hardware I/O: Model B60 with Three I/O Buses

completion of I/O requests. Data is moved by I/O controllers, which access System Processor main storage as DMA masters while the System Processor's bus adapter functions as DMA slave.

Bus Fault Detection

The I/O bus was designed to minimize the disruption caused by the failure of single bus unit. Each I/O bus unit contains facilities to detect, identify, and recover from failures. A time out occurs when an I/O bus unit detects a bus failure and suspends bus operation. The bus control unit detects the time out and causes each I/O bus unit to store pertinent error information into a status register. In this way, hardware in each I/O bus unit, including those not involved in the failing operation, is an independent monitor of bus failures. The status captured at the time of failure permits isolation of intermittent and solid bus failures.

The I/O bus unit involved in the time out enters a disabled state and is unable to participate in subsequent normal bus operations. This mechanism prevents failing I/O bus units from disrupting communications between the System Processor and other I/O bus units for many bus failures.

When a time out occurs, software in the System Processor uses special bus commands (only available to the bus control unit) to collect status from all the bus units on a bus. The collection of this status is transparent to software in the I/O controller; consequently, operations in controllers not involved in the time out are unaware of the failure and recovery activity. This collected status is used to identify which I/O bus unit caused the failure. The disabled I/O bus unit is then either enabled to try the failing operation again or left disabled until repaired.

Software Structure for Input/Output

One of the capabilities introduced with the I/O bus in the AS/400 system is the ability for an I/O bus unit to send unsolicited work requests to the System Processor. To facilitate this capability, a process-to-process programming mechanism was designed (see Figure 2). The process-to-process programming mechanism provides symmetrical flows between the System Processor and the I/O controllers on the I/O bus. With symmetrical flows, the I/O bus units and the System Processor have the same functional capability to initiate and perform work.

A program's interface to the I/O bus has been defined in terms of a set of verbs. Verbs are the commands or functions that the process-to-process mechanism can perform. For example, RECEIVE DATA and SEND REQUEST are verbs. Communications between processes is in terms of sending and receiving messages and data over a logical connection between them. A layer of code in each I/O bus unit, referred to as the bus manager, determines the capabilities of each unit it communicates with (slave DMA, master DMA, or both) and controls the flow of messages and data across the I/O bus.

For the bus managers in the System Processor and an I/O controller to communicate, they need:

- Bus messages: Fixed-length control information to be transferred from one bus unit to another. Two principle messages are OP-START and OP-END.

- Request-response control block: Controls the movement of data, commands, and control information between the requestor and server.

Work is started in process B (the server) when a request is presented at the verb interface (see Figure 2) by process A (the requester). Because process B is in a different processing unit than

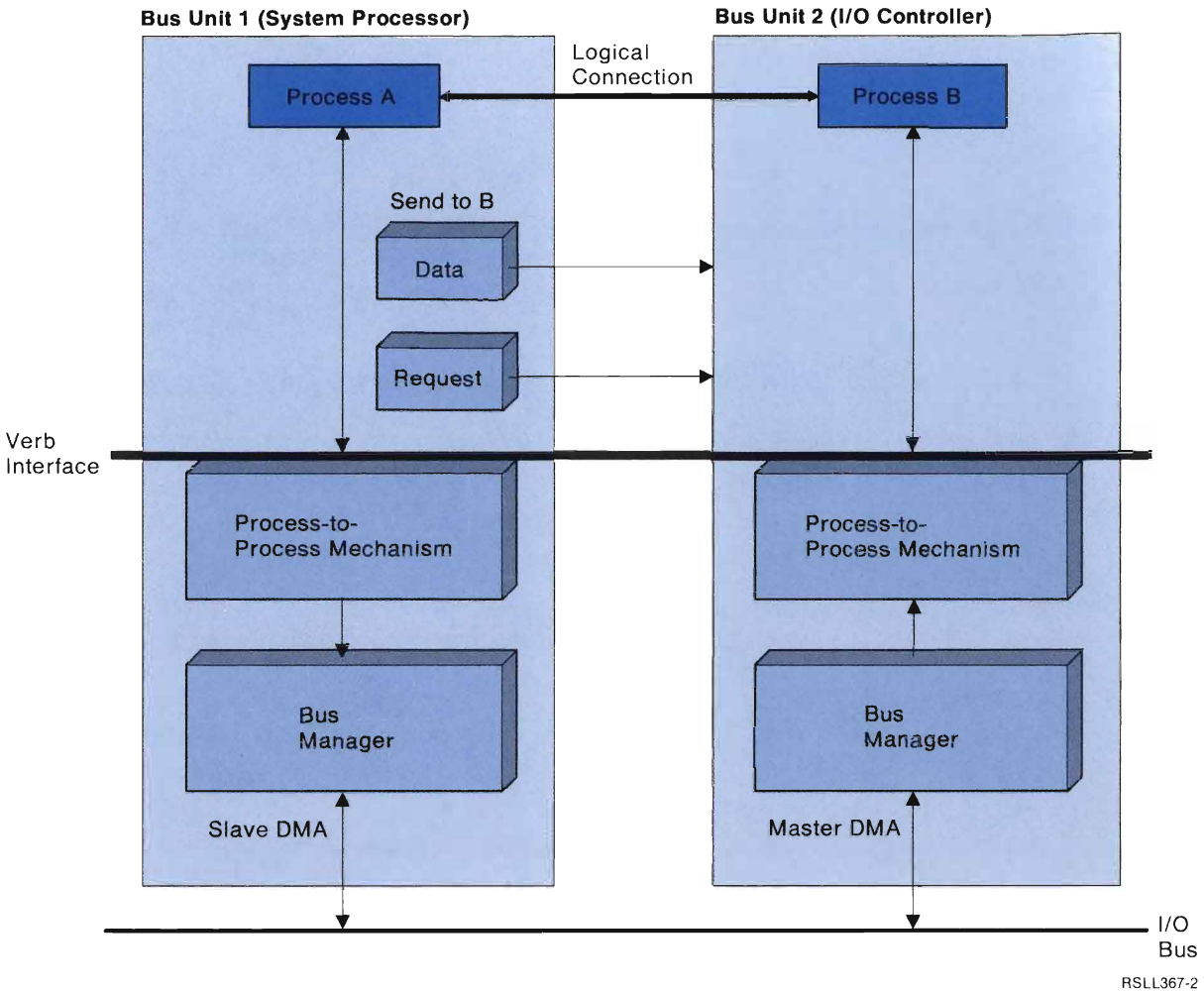


Figure 2 Process-to-Process Mechanism and Bus Manager, Normal Flow

process A, the unit 1 bus manager builds a control block, the request-response control block, and sends an OP-START bus message to alert the unit 2 bus manager that a request is pending. The OP-START bus message has sufficient information for the unit 2 bus manager to move a copy of the request-response control block into bus unit 2. The bus manager, through the unit 2 process-to-process mechanism, can now alert process B that work is to be done. Process B transfers the data between the bus units. The pacing of data transfers is controlled by bus unit 2. Unit 2's copy of the request-response control block is used by its bus manager to control the transfer of data between the bus units. Bus unit 2 signals completion of the request by sending an OP-END bus message to bus unit 1. Process A is notified by its bus manager when the OP-END is received.

In the AS/400 system, I/O bus-attached I/O controllers do not have slave DMA capability. Therefore, to maintain symmetry of the data

movement at the process-to-process interface, an additional data transfer method, reverse flow, is supported by the bus manager.

With reverse flow, it is possible for an I/O controller (I/O bus unit) with only master DMA to request an operation from a server process with only slave DMA capability. In this instance, a pool of buffers in System Processor storage (the bus unit with slave DMA) is available to the bus manager in an I/O controller (the bus unit with master DMA and containing the requester process). The System Processor's bus manager controls the number of buffers available to the I/O control unit. The I/O control unit uses the buffers as required. Work is initiated at the process interface and the bus manager is started as before. The bus manager in bus unit 2 (see Figure 2) realizes the asymmetric DMA capabilities and builds the request-response control block, then moves the request-response control block and data, using DMA, directly into the remote buffer storage in the System Processor that was allocated for I/O bus unit use. At this point, the bus manager in unit 2 sends an OP-START bus message to unit 1 to notify it that a request is pending. The server process in the System Processor requests data through the process-to-process mechanism, which results in data being moved from one location (the buffer) to another within the Processor's storage. When the Processor has completed the request, an OP-END bus message is sent to the I/O control unit indicating the operation is complete. The same bus messages and control block are used, although the underlying hardware support is not the same.

Conclusions

The I/O structure of the AS/400 system is based on a new 32-bit I/O bus. The I/O bus supports a bus control unit and up to 31 additional, independent I/O bus units. The bus is designed to provide flexible I/O configuration and expansion, and intermittent and solid I/O bus fault detection. System models with one, two, or three I/O buses are supported. In addition, a process-to-process communications mechanism has been designed such that the System Processor and the I/O bus units have the same functional capability to initiate and perform work. This functionality, together with newly designed I/O controllers, offers I/O functional capabilities normally associated with much larger systems.

Acknowledgements

The authors would like to thank Richard A. Kelley, IBM Boca Raton, FL, for his contributions to the definition and documentation of the bus architecture. Thanks also to Richard E. Zelenski, IBM, Rochester, MN, for the guidance he provided in the early stages of system I/O definition.

Magnetic Storage Device Controller

Discusses the unique microcode design which optimizes performance while maintaining concurrent operations between multiple devices.

Fred L. Huss, Gene A. Lushinsky, Kevin P. Gibson, and Surinder P. Batra

Introduction

The Magnetic Storage Device Controller, used in AS/400™ 9406 System Units, provides the control and data transfer path between the system input/output (I/O) bus and the IPI-3 bus.

A single device type, or a combination of disk, tape, and diskette units, can be attached to the Storage Device Controller, which must maximize the data transfer rate between the device and the 9406 System Processor, while managing concurrent operations with multiple devices (for example, reading data from disk unit 1 while writing data to disk unit 2). Providing concurrent device support, combined with varied device performance characteristics, requires three solutions. First, the time a device is idle must be minimized, so the Storage Device Controller must provide parallel processing. It does that by minimizing the sequential processing time with each device. Also, as the number of system operations increase, the microcode must minimize the effect of increasing device and controller usage. According to queuing theory, run time increases by a factor of one divided by one minus the utilization (1 / (1-U)). As the utilization increases, the run time increases dramatically and must be reduced. The microcode operation minimizes the use of the controller and the device while processing heavy I/O loads. And, finally, commands and data transfers to the devices must be started and continued on a timely basis. Unique device timing characteristics require prompt data transfers to prevent extra disk revolutions or tape backhitches (a brief rewind and restart). These key performance requirements were met in the innovative design of the Storage Device Controller.

Hardware Structure

The hardware provides separate direct memory access (DMA) and control processor buses that allow the bus interface hardware to operate independently of the control processor (see Figure 1). For read or write data transfer operations, the control processor sets up the system adapter, device adapter, and DMA controller to provide the data path between system storage and the device. The DMA hardware controls the data transfer but periodically interrupts the control processor to continue or complete the data transfer. Maximum throughput is obtained during a write, for example, by allowing the device adapter to empty DMA storage while the system adapter fills it.

Microcode Structure

The microcode consists of two priority interrupt-service routines and three tasks (referred to as ISR in Figure 2). Work items are placed on a task's queue by other tasks or by one of the interrupt-service routines. The control program activates a task if a work item is on the queue and the priority of that task is higher than any other tasks with work items queued.

The system and IPI-3 bus managers are interrupt-service routines that support the bus adapter hardware. The interrupts signal the microcode that the System Processor has an I/O request, that a device is ready to service an I/O request, or that a DMA transfer has completed.

The primary task is the device manager, which translates system storage I/O requests into IPI-3 bus protocol and manages concurrent device activity on the IPI-3 bus. The services connection manager and the reliability and serviceability (RAS) manager are the two other task functions that handle logical connections used for communications, diagnostics, error logging, and configuration functions of the controller and the devices.

Performance Implementation

Solving the performance problems while still providing concurrent device support (automatic multiplexing between active disk, tape, and diskette devices) requires a unique controller microcode solution: two interrupt-service routines and the device-manager task minimize synchronous time, as shown in Figure 3. The unique design allows the interrupt-service routines to share device queues and data structures with each other and with the device manager task. In addition, the next device command is initiated from the interrupt-service routine before completing the active operation (shown as IOP Microcode Sequence in Figure 3).

The overall response time is the most important subsystem performance criteria as measured on

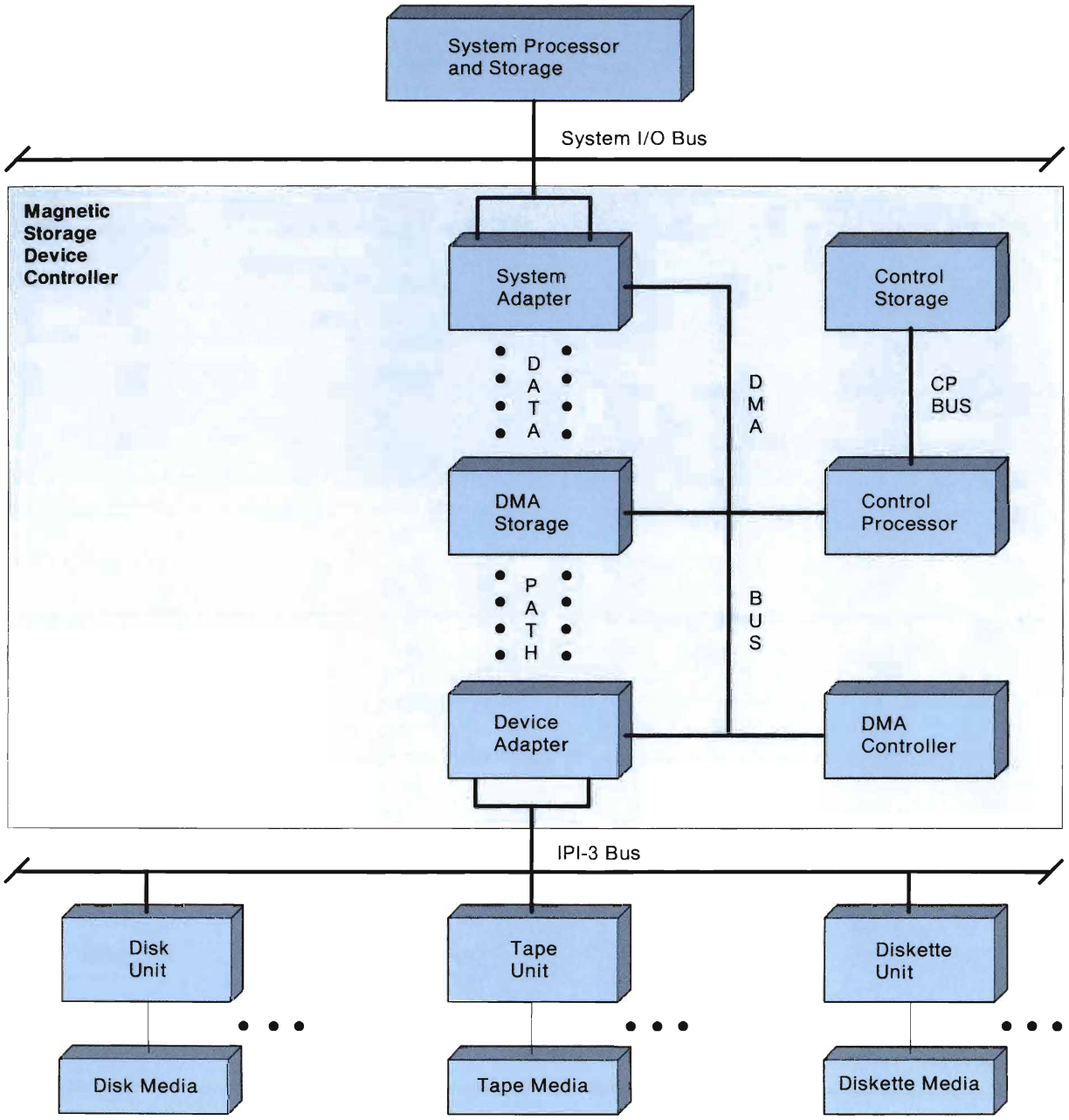


Figure 1 Magnetic Storage Device Controller Functional Block Diagram

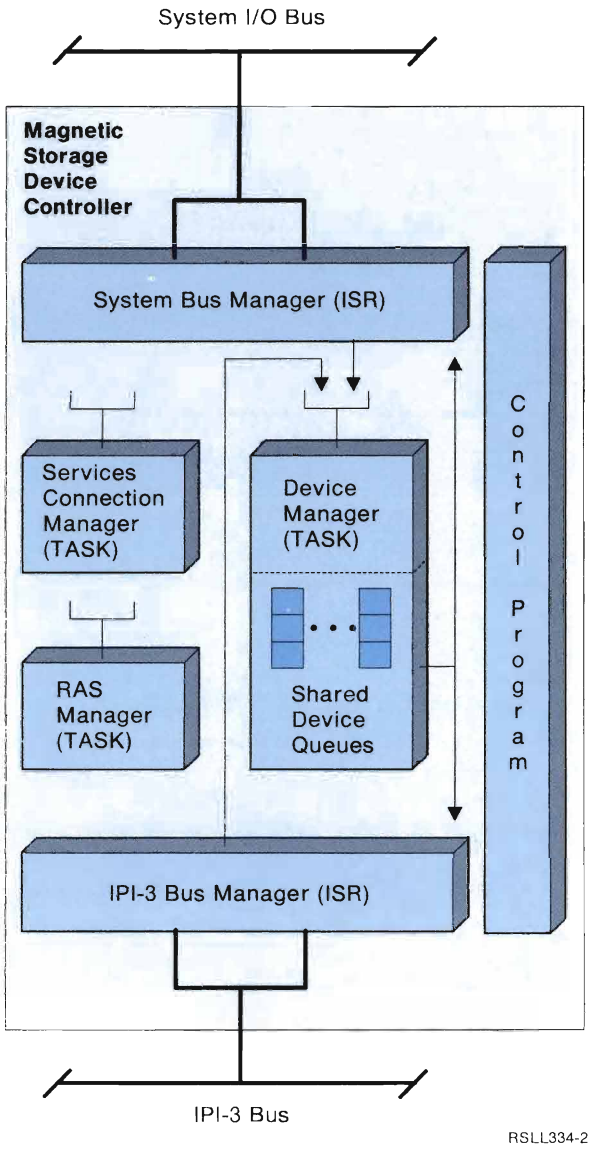


Figure 2 Magnetic Storage Device Controller Microcode Overview

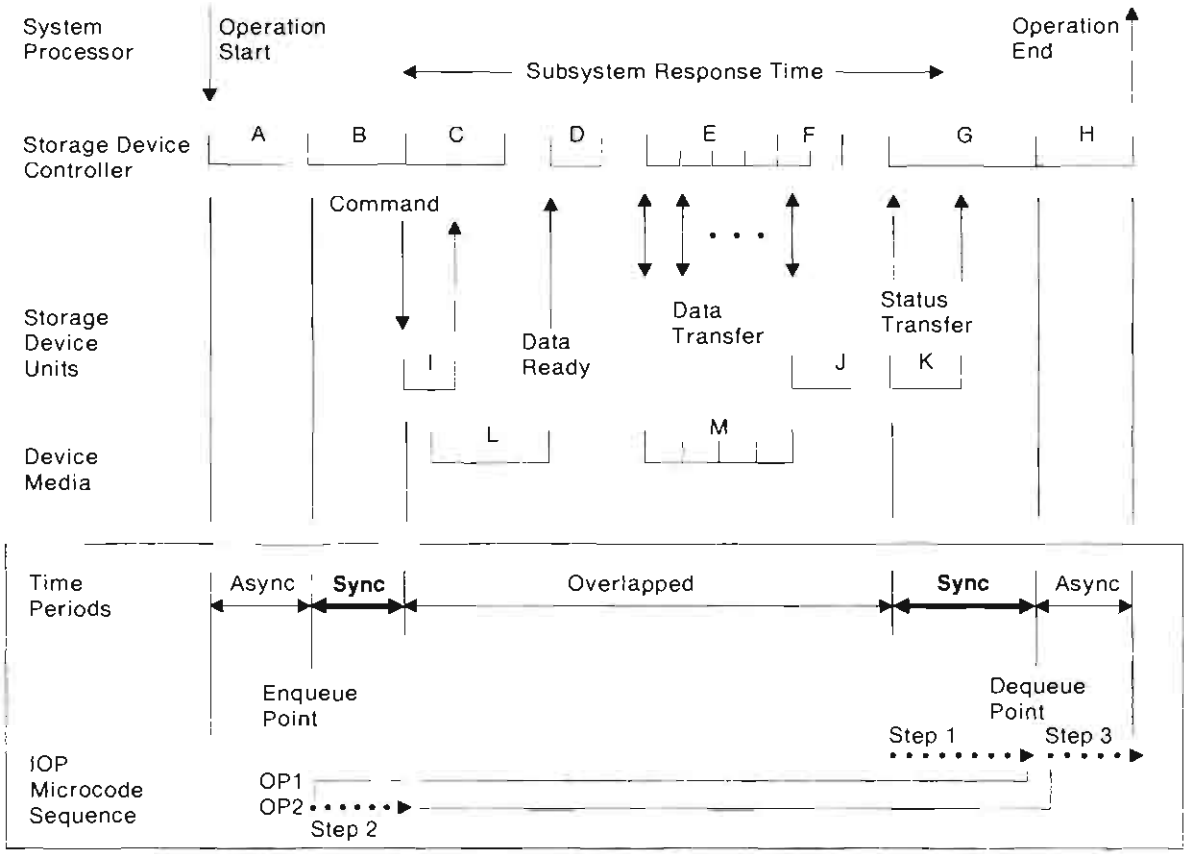


Figure 3 Magnetic Storage Device Controller Performance Time Periods

the system I/O bus from the start to the end of the operation. Several performance time periods must be optimized to obtain the best performance. These time periods are illustrated in Figure 3, where the concepts of asynchronous, synchronous, and overlapped times and the enqueue and dequeue points are shown. These are defined as:

- Asynchronous time: Both the controller and device are active.
- Synchronous time: The device is waiting on the controller.
- Overlapped time: The controller is waiting on the device (the controller is free to service other operations).
- Enqueue point: The point where a new operation must wait for a current operation to the same device to complete.
- Dequeue point: The point in the current operation where an enqueued operation to the same device is allowed to continue. The current operation is temporarily suspended while the controller dequeues the waiting operation.

AS/400 applications and system programs often access data that is physically stored on the same media. The system forwards these operations to the controller and it must decide whether to process them immediately or queue them temporarily while waiting for a previous operation to complete. As the AS/400 system becomes heavily loaded, the queuing of requests in the controller becomes more frequent and then synchronous time becomes more important to performance.

For the subsystem to avoid extra disk revolutions and maintain streaming tape operations, the same key microcode design requirement to minimize synchronous time applies. To avoid an extra disk revolution or to prevent a tape backhitch, the device must be sent the next command within a very short time (synchronous time).

Overlapped microcode time is best for performance, because the controller is waiting for the device and may service operations to other I/O devices. The asynchronous time period is the next best; though the controller time adds to the subsystem response time, the device is not waiting for the controller, even under queued

conditions. Synchronous time is the least desirable situation because, under queued conditions, the device is waiting for the controller, unproductively increasing the device utilization.

The microcode design minimizes synchronous time by moving function from the synchronous period into the asynchronous period. When the current operation finishes data transfer, it is temporarily suspended while a new operation is dequeued and the device is restarted. Once the device has been restarted with the new operation, the microcode resumes processing of the temporarily suspended operation. The function of dequeuing the new operation is processed in interrupt context, rather than task context, to minimize this time. When a new operation is received and must be enqueued because the current operation to the same device is not complete, the microcode does as much pre-processing of this new operation as possible. In addition, an algorithm sequences the enqueued operations to minimize the disk-seek distance, and thus the seek time, as the operations are dequeued.

Allowing interrupt-service routines and the device-manager task to share common functions and control blocks requires a design that matches device characteristics to the controller microcode function. To maintain the flow of data from a device and minimize synchronous time, the microcode gives priority to device-adapter interrupts. This is accomplished by forcing a lower-level, internal-microcode interrupt for the system adapter, thus preventing multiple back-to-back system interrupts from locking out device interrupts for long periods of time. The system-adapter interrupt-service routine also allows device-adapter interrupts to be processed before it has finished; this is especially important for tape streaming. In addition, suspending the op-end processing (step 1 in Figure 3) while initiating a new operation (step 2), and then resuming the op-end processing (step 3) requires innovative control techniques in the interrupt-service routines.

Conclusions

The AS/400 Magnetic Storage Device Controller solves the performance problem of maximizing the data transfer rate between the device and the System Processor while maintaining concurrent operations between multiple disk, tape, and diskette devices. This is achieved by a unique controller microcode design of interrupt-service routines and task microcode that minimizes synchronous time, which is the key subsystem-performance time period.

Work Station Controllers

Discusses the functions provided by AS/400 work station controllers and describes the microcode and hardware structure developed to implement those functions.

Jeffrey E. Remfert, Trent L. Clausen, Gregory A. Dancker, and Harvey G. Kiel

Introduction

AS/400™ work station controllers provide a cost-effective means for attaching display stations and printers to the system by supporting a wide variety of synchronous and asynchronous display station and printer devices (see Figure 1). Display station screens range from 24 lines by 80 columns to 27 lines by 132 columns. Printer speeds range from 40 characters per second to 2000 lines per minute. In addition, the IBM Personal Computers and Personal System/2™ family can be attached for use as programmable work stations.

The primary function of the AS/400 work station controllers is to perform data stream and keystroke processing for attached display stations. Additionally, the controllers provide protocol-conversion support for ASCII printers and data stream pass-through mechanisms for synchronous printers and attached personal computers. The distribution of data stream and keystroke processing frees the host system for application processing and allows attachment of cost-effective display stations. The display station and printer data stream support provided by the AS/400 work station controllers facilitates System/36 and System/38 application program portability.

The work station controllers can be connected either locally to the AS/400 input/output (I/O) bus or remotely to the AS/400 data communications subsystem. AS/400 work station controller enhancements include: support for directly attaching asynchronous (ASCII) display stations and printers; improved functional transparency between local and remote work stations; integrated national language support; and improved word processing support.

Key work station controller design objectives were to: present a common operating system interface; provide functional transparency between local and remote controllers; integrate national language support; and provide highly reliable, easy-to-service hardware and microcode. A layered microcode structure and highly integrated hardware logic were used to develop the family of controllers. The layered-microcode design approach minimized development effort and provided functional consistency between the controllers. Extensive performance modeling was used to help make design decisions.

Layered Microcode Structure and Function

The microcode functional layers, shown in Figure 2, are organized into three major groups:

1. Host-system attachment components
2. Common-function components
3. Device-attachment components

The microcode components can be bound together to form the following controllers: a local synchronous controller is formed by binding components (in the figure, connected using ----), a local asynchronous controller is formed by binding components (connected using ·····), and a remote synchronous controller is formed by binding components (connected using · · ·).

Host-System Attachment Components

The first group of layered microcode is the host-system attachment interface, consisting of two types: a system I/O bus interface for a local controller and a data communications interface for a remote controller. The I/O bus interface component uses the AS/400 I/O bus protocols to communicate with the host processor. Bus unit messages are used to transfer control information and direct memory access (DMA) is used for transferring data. The data communications interface for a remote controller uses synchronous data link control (SDLC), X.21, or X.25 protocols to communicate with the host system across a telecommunications line.

Common-Function Components

The next group of layered microcode is the common-function layer used by both local and remote controllers. The Systems Network Architecture (SNA) component, supporting LU-LU session types 4 and 7, is used to establish, maintain, and end sessions between the user application programs and the attached display stations and printers. The SNA component provides a mechanism for transporting user data streams between the host system and the controller, and facilitates functional transparency between local and remote controllers. Data streams supported are those defined for the 5250 Information Display system. To support display stations, the controller data stream and keystroke

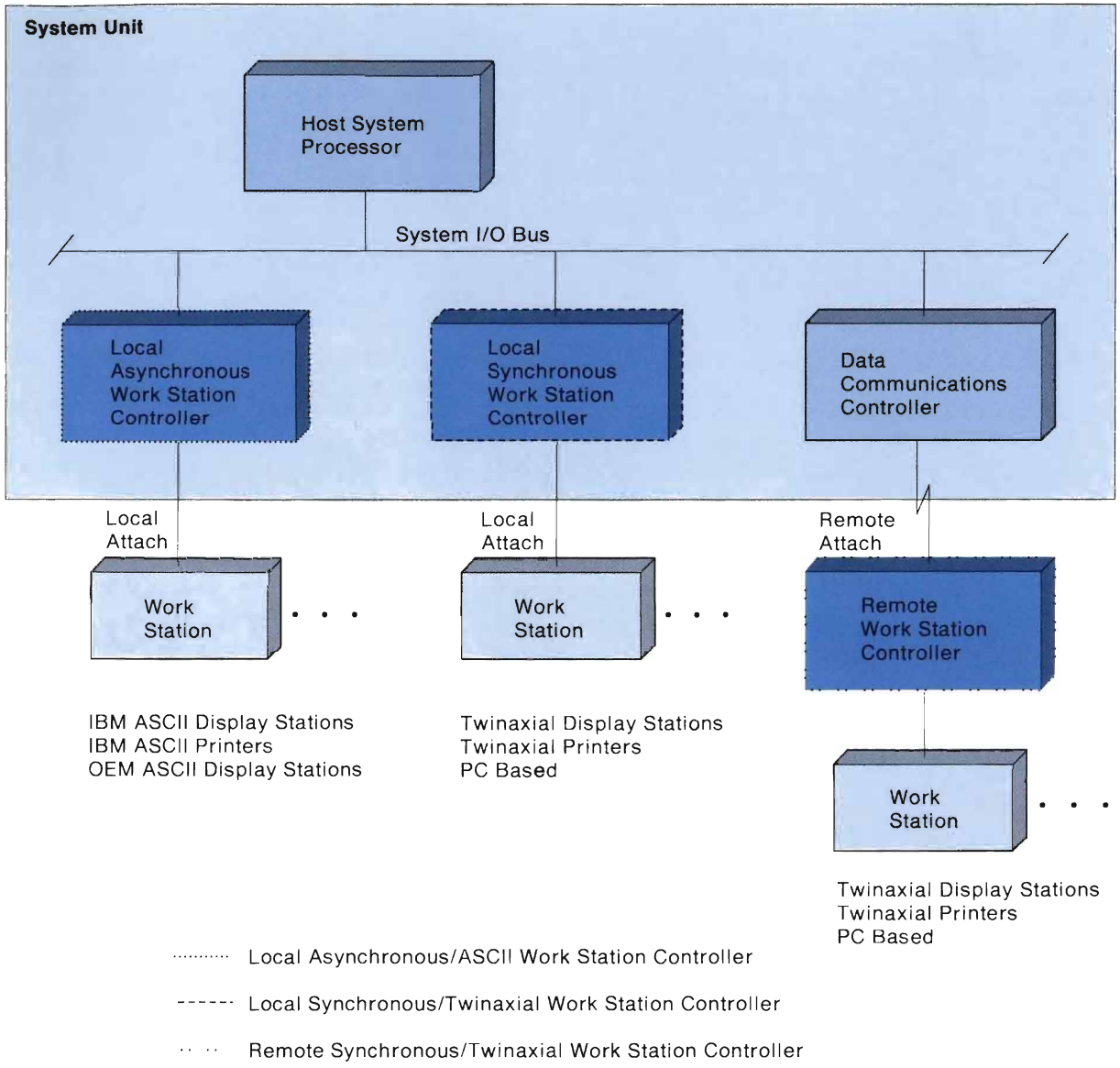


Figure 1 Work Station Controller Subsystem Overview

components work together to process the user data stream and control the display station. The data stream contains commands and orders that tell the controller how to format the display data and define input-field edit characteristics. AS/400 work station controllers are editing controllers, meaning the controller validates the data entered by the work station operator based on each input field's characteristics. The data stream component handles PUT and GET operations, which write information to the display station and read information from the display station. The keystroke component processes display station keystrokes. When a key is pressed, the keystroke component receives the keyboard scan code from the display station, translates it into a character, then writes the character to the display station screen. The translation process involves a unique translation table for each type of keyboard supported. Keyboard types are based on keyboard style (layout) and national language.

National language support is an integral part of AS/400 work station controllers. The national language support provided by the synchronous controllers is divided into three groups. (The asynchronous controllers support a subset of these groups.)

- Two-shift keyboard and left-to-right display support
- Four-shift keyboard and left-to-right display support
- Four-shift keyboard and bidirectional display support

Countries using the two-shift keyboard and left-to-right support have languages that require two layers of characters on their keyboards. Character and field directions are from left to right. Examples of languages in this group are English, French, German, Italian, and Japanese Katakana.

Languages requiring four-shift keyboard, left-to-right support need four layers of characters on their keyboards, where the selection of a character involves shifting into the appropriate layer for the desired character. Character and field directions are from left to right. Examples of languages supported in this group are Cyrillic, Greek, and Thai.

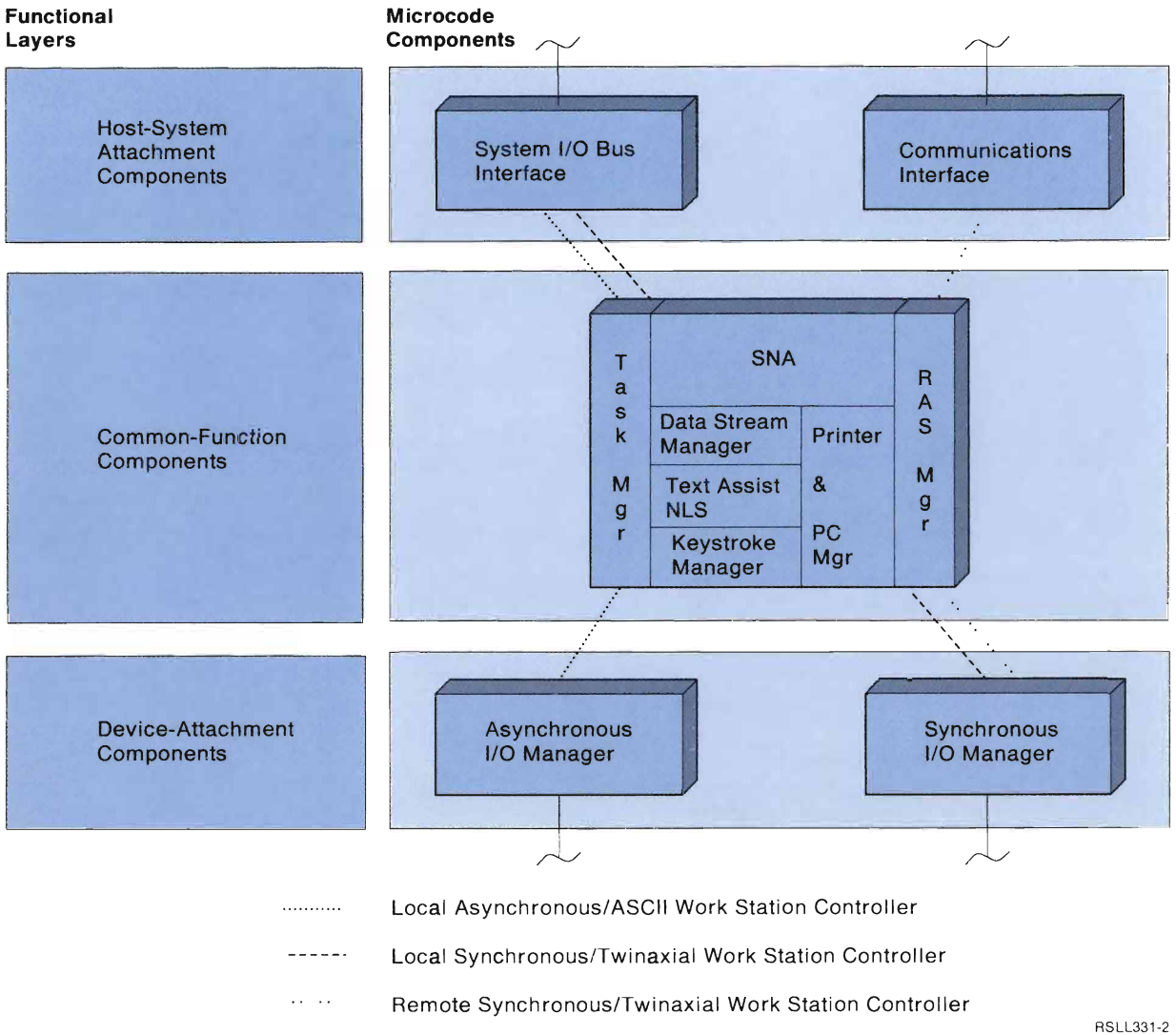


Figure 2 Functional Overview of Layered Microcode Components

For languages requiring four-shift keyboard and bidirectional support, character and field directions can be right to left or left to right in any combination. Both character and field directions are specified by the application program. In addition to the right-to-left data entry capability, support for the Arabic language includes an automatic character-shape determination function. In Arabic script, the shape of each character depends upon its position within the word. Automatic character-shape determination facilitates the display of Arabic script by shaping each character as it is typed. Arabic and Hebrew are the languages supported in this group.

In addition to the basic display functions, the AS/400 work station controllers provide word processing functions, including: word wrap and continuous insert (gives the user the appearance of an infinitely-long sheet of typing paper); scale line (shows tab stops and margin positions); copy, move, and delete capability (on a block, line, or word basis); center-text capability; word underscore; and split-screen capability. The distribution of function between the software in the host processor and the work station controller is tightly coupled to offer optimum performance. Host-processor interruptions due to function keys are kept to a minimum.

In addition to display stations, the work station controllers support printer devices and attached personal computers. For synchronous printers, the data stream commands and orders are passed through the controller to the printer where they are interpreted; for asynchronous printers, the controller emulates the data stream commands and orders received from the host processor using ASCII printer commands. The commands and orders perform various printer control functions, such as formatting the data and starting a new line or page. Attached personal computers support functions such as 5250 display station emulation, file transfer, virtual disk, and virtual print. A streamlined data transfer mechanism between the controller and the attached personal computer was developed to provide optimum performance.

Completing the set of common functions are the task manager and the reliability, availability, and serviceability (RAS) support. The task manager, or control program, manages all of the activity in the controller. Task control blocks as well as task priorities, work queues, and a storage allocation mechanism allow the functional components to communicate with one another and control the transfer of data. Additionally, significant effort was spent in the early stages of development to

ensure reliability, availability, and serviceability of the controllers. Error retry and logging are an integral part of the design, as are built-in tools for servicing the microcode. Service tools such as read/write controller storage, set dynamic trace points, and task control-block trace are provided. The reliability, availability, and serviceability component also collects performance measurement data and returns it to the host processor.

Device-Attachment Components

The third major group of layered microcode is the device attachment interface, which consists of two types: a synchronous I/O manager component for the synchronous controllers, and an asynchronous I/O manager component for the asynchronous controller. The data stream and keystroke components send requests to the synchronous or asynchronous I/O managers. The synchronous I/O manager interprets the requests and generates control blocks (in controller storage) for the synchronous hardware adapter, to transfer commands and data to or from the attached device (see Figure 3).

Before the asynchronous I/O manager sets up control blocks for the asynchronous I/O adapter hardware, a protocol conversion from 5250 display and printer data streams to ASCII data streams is performed. The requests from the data stream and keystroke components are used to generate and maintain a display image, in controller storage, for the target device. That display image is then interpreted by the protocol conversion program, which generates the appropriate ASCII data stream for the target device. To optimize performance, microcode algorithms were developed to minimize the amount of data sent to an ASCII display station (for example, only the updated or changed portions of a display are sent to the display station). The protocol conversion is table-driven; attribute and keyboard mapping tables are maintained for each

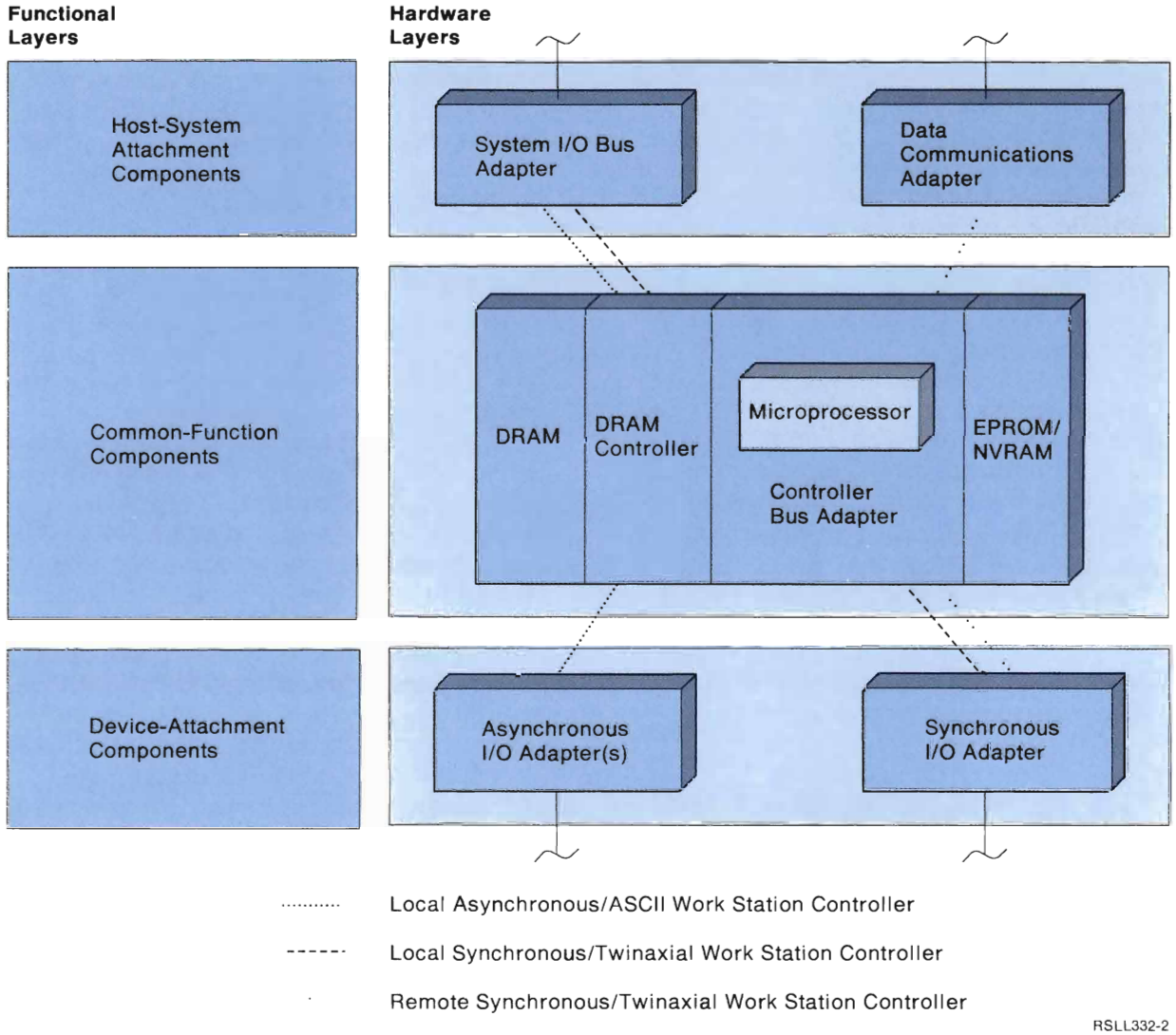


Figure 3 Functional Overview of Layered Hardware Components

type of device. Also, in the conversion process, an EBCDIC-to-ASCII data translation is performed. When the protocol conversion is complete and the control blocks and data are set up in storage, the asynchronous I/O adapter hardware is activated to transfer the commands and data to or from the device.

Integrated Hardware Structure and Function

AS/400 work station controllers are microprocessor-based. The overall hardware structure is similar to the microcode structure, as shown in Figure 3. As with the microcode components, the hardware components (host-system attachment, common-function, and

device- attachment interfaces) can be combined to form any of the controllers.

Host-System Attachment Components
The first group of hardware layers is the host-system attachment interface. This can be one of two types: an AS/400 I/O bus interface for local work station controllers or a data communications interface for a remote controller. The AS/400 I/O bus adapter supports DMA operations and allows bus unit messages to be transferred to and from the host processor.

The data communications interface consists of a module that provides DMA to and from controller storage, plus an adapter that provides standard telecommunications electrical and physical interfaces. This hardware is directed by control blocks set up in work station controller storage. Under microprocessor control, registers are initialized in the hardware, a particular command is issued, and the communications-interface hardware processes the request, freeing the microprocessor for other tasks.

Common-Function Components
The second group of hardware components provides functions that are common to the remote and local controllers. The microprocessor bus interface, bus arbitration, and interrupt-priority logic are provided in this layer of hardware. The local controllers use up to one megabyte of dynamic random access memory (RAM) for control and data storage. The dynamic RAM controller provides read and write control, refresh control, and single- and double-bit error detection. The erasable programmable read-only memory (EPROM) is used for initial microcode load (IML) and diagnostics. The non-volatile RAM is used for vital product data (part number, serial number, and plant of manufacture). The remote controller uses a combination of read-only storage (ROS) and dynamic RAM for instruction storage and dynamic RAM for data storage. Logic is also provided for the device-attachment components. This logic provides DMA and interrupt capability for the device-attachment components and allows the microprocessor to write and read registers in the device-attachment components.

Device-Attachment Components
The third hardware group is the device-attachment interface. The synchronous I/O adapter is capable of driving up to eight ports, with a capacity of seven work stations per port. It is a command-driven controller that operates on control blocks residing in the dynamic RAM. It has internal DMA capability and can address one megabyte of controller storage. The synchronous I/O adapter has two control-block chains (the automatic poll and I/O chains) as well as two timers. The chains are formed by linking individual control blocks together. This allows the synchronous I/O adapter to sequence through a string of control blocks when a single Start Automatic Poll or Start I/O command is issued to it. These functions are available to the microcode and are used by loading the internal control registers. The synchronous I/O adapter handles all work station polling on the automatic poll chain and requires little processor intervention after the chain is set up in controller storage. Only when a keyboard scan code returns or an error occurs will an interrupt be posted. When this happens, the synchronous adapter stops processing the control block on following cycles of the automatic poll chain. The I/O chain and associated timer are used for transmitting large blocks of data to and from the work stations.

The synchronous I/O adapter was designed to be used with a balanced-line driver/receiver module. A single module provides the eight ports.

The asynchronous I/O adapter consists of up to three asynchronous I/O modules, with each module capable of supporting six asynchronous devices. This allows the controller to support up to 18 asynchronous devices. Devices can be attached locally, or, using a modem, remotely. A universal asynchronous receiver/transmitter (UART) is provided for each device port. The UARTs operate independently, so operating characteristics, such as data length, line speed, and the number of stop bits, can be individually configured. Each UART contains a 2-byte receive buffer and a 2-byte transmit buffer. Data transfer between a UART and controller storage can be one of two modes of operation. In the first, a byte-transfer mode, the UART generates an interrupt to the microprocessor whenever its receive buffer is full or its transmit buffer is empty. The microcode then reads the receive buffer or writes to the transmit buffer to send the data. The second, block data transfer, uses DMA. Each UART is assigned an 8-byte control block in controller storage. The control block contains the current and ending receive and transmit addresses. The UART requests a DMA transfer whenever its receive buffer is full or its transmit buffer is empty. The DMA controller accesses the associated UART control block and transfers the data. When the ending address is reached, the block data transfer is complete and the DMA controller interrupts the microprocessor to indicate completion status. The block data-transfer mode of operation requires minimal microprocessor involvement, thus freeing the microprocessor for other tasks.

Performance Characteristics
Work station subsystem performance is dependent on the characteristics of the work station controller hardware and microcode, the attached devices, and the attachment media. As described, the synchronous and asynchronous I/O hardware adapters are highly functional, reducing the load on the microprocessor. The synchronous I/O adapter does the device polling for keystrokes and device busy, and both synchronous and asynchronous I/O adapters perform block data and command transfers to or from controller storage and interrupt the microprocessor when

the transfer has completed. The work station controller microcode is multithreaded, which means that the work station controller and attached-device processing are overlapped. For example, when the controller has sent commands and data to a device, it then starts to process the data stream for the next device. This characteristic allows the controller to maintain a high level of performance, even as additional devices are added.

During the development of the work station controllers, extensive performance modeling was done and measurements were taken. Typical work station controller work loads were characterized. Performance capacity limits for each work load were studied and service times were optimized. Display station work loads were characterized as data processing (interactive commercial application) and word processing (office application). A data processing work load emphasizes PUT and GET processing, while a word processing work load is predominantly keystroke processing. This necessitated interleaving PUT and keystroke I/O processing to give balanced performance. Printer work loads were used to verify that printers were driven at their rated speeds. Attached personal computer work loads were used to simulate file transfer activity.

Additionally, local work station controllers collect performance measurement data, such as: microprocessor utilization, device-attachment I/O adapter utilization, task manager queue length counts, and display station response time (the time from when the operator presses the Enter key until the work station controller unlocks the keyboard). Each display station's response time is kept to indicate the actual response time experienced by the user. A host-processor performance monitor retrieves the performance measurement data from the work station controller, formats the data, and then presents a summary of the data to the system operator. The information presented can indicate where system work load balancing or configuration adjustments are needed.

Conclusions

AS/400 work station controllers allow the attachment of a broad range of work station devices. The controllers provide a high level of function, including field editing, keystroke processing, word processing features, and national language support, that relieves the host-processor load.

The key work station controller design objectives were to present a common operating system interface, provide functional transparency between local and remote controllers, integrate national language support, and provide highly reliable, easy-to-service hardware and microcode. The use of a layered microcode structure and common, highly integrated hardware logic facilitated the development of a family of work station controllers that meet these objectives.

The Multiple-Function Input/Output Processor

Describes the capabilities of the Multiple-Function Input/Output Processor, the design philosophy, and the hardware and microcode technologies used.

Charles A. Lemaire, Renato J. Recio, and Stephen P. Hank

Introduction

The AS/400™ Multiple-Function Input/Output (I/O) Processor was developed to meet the needs of the AS/400 9404 System Unit. The Multiple-Function I/O Processor, a combination of hardware and microcode, merges the functions for a service processor, magnetic-media storage device control, and communications control into a one-card I/O processor. These functions were previously implemented by three separate I/O processors. The design is a significant breakthrough in minimizing product costs of the 9404 System Unit. The Multiple-Function I/O Processor gives small system models the full-speed performance of the attached I/O devices, with a minimum of I/O processor overhead. The design is flexible, allowing implementations that provide dedicated magnetic media and communications I/O processors to be easily derived from the primary multiple-function design. This approach allowed the best of existing I/O processor designs to be combined into fewer cards, while allowing incremental performance improvements in specific functions through the dedicated I/O processor cards derived from the same design.

Design Philosophy

The Multiple-Function I/O Processor is a combination of hardware and microcode that performs low-level control of disk devices or communications lines, combining the **service processor, media storage device control**, and **communications control**. Commands sent by the 9404 System Processor specify the various operations to be performed by each I/O processor in the system.

The service processor function performs the initial program load (IPL) for the system, provides an interface to the customer control panel, and diagnoses the system I/O bus when the System Processor cannot. The service processor also has a time-of-day clock supporting a timed power-on function for the system. Special storage contains the system vital product data, which has numbers to identify part type, engineering change level, and serial number.

The magnetic-media processor function services disk, tape, and diskette I/O requests from the System Processor, controls the magnetic media devices and data flow, and analyzes errors. Two interfaces facilitate attachment of magnetic media devices. A Small Computer System Interface (SCSI) bus provides the interface for disk and tape devices. This SCSI bus is asynchronous and supports a 1.5 megabyte-per-second data rate. An ANSI 3.8 interface allows the attachment of either a 5.25- or 8-inch diskette drive.

The communications processor function handles data and commands to and from the communications lines. The microcode supports four communications protocols: asynchronous, binary synchronous (BSC), synchronous data link control (SDLC), X.21, and X.25. Three electrical interfaces are supported: RS232, X.21, and V.35; each of these is implemented on a separate small book communications card.

In the process of combining the three functions into one card, the system cost is reduced by eliminating redundant hardware parts and using common code routines. The hardware of a single microprocessor, control storage and storage controller, and system I/O bus adapter is time-shared by the various I/O processor functions. The control program, bus interface code, and other common code have just one version for both the Multiple-Function I/O Processor and the single-function cards derived from it. On the multiple-function card, these programs appear in main storage just once but service several I/O functions, thus reducing overall storage costs.

The Multiple-Function I/O Processor was designed to be fast enough for the system to benefit from the full speed of the attached I/O devices. Dedicated, single-function I/O processors were then derived by depopulating the basic design. These single-function I/O processors have better performance than the Multiple-Function I/O Processor because they do not have to spend time switching between the various functions. The design team was able to maximize the performance of each I/O processor by optimizing one basic design. The 9404 System Units can be configured and fully functional with just one Multiple-Function I/O Processor, an internal microprogramming interface (IMPI) card, and a work station controller. The Multiple-Function I/O Processor card provides small models with competitive I/O processor performance. Incremental performance improvement in particular areas can later be

obtained by adding dedicated-function cards as needs arise or budgets allow (see Figure 1).

Performance improvements over other dedicated single-function I/O processors were needed to meet system performance objectives when all three functions were performed from one I/O processor. Enhancements created for the Multiple-Function I/O Processor consist of multiple data paths (from devices or communications lines to the system) through the I/O processor, and an embellished microcode structure to make use of the new functions. This improved software structure provides the utmost quality because a single hardware design is made to work, and work well, for all three of the major functions.

The Multiple-Function I/O Processor is flexible enough for dedicated magnetic media and communications I/O processors to be derived from the primary multiple-function design. The hardware chip set from a single design effort can be used as a **Multiple-Function I/O Processor**, a **communications I/O processor**, or a **magnetic-media I/O processor**.

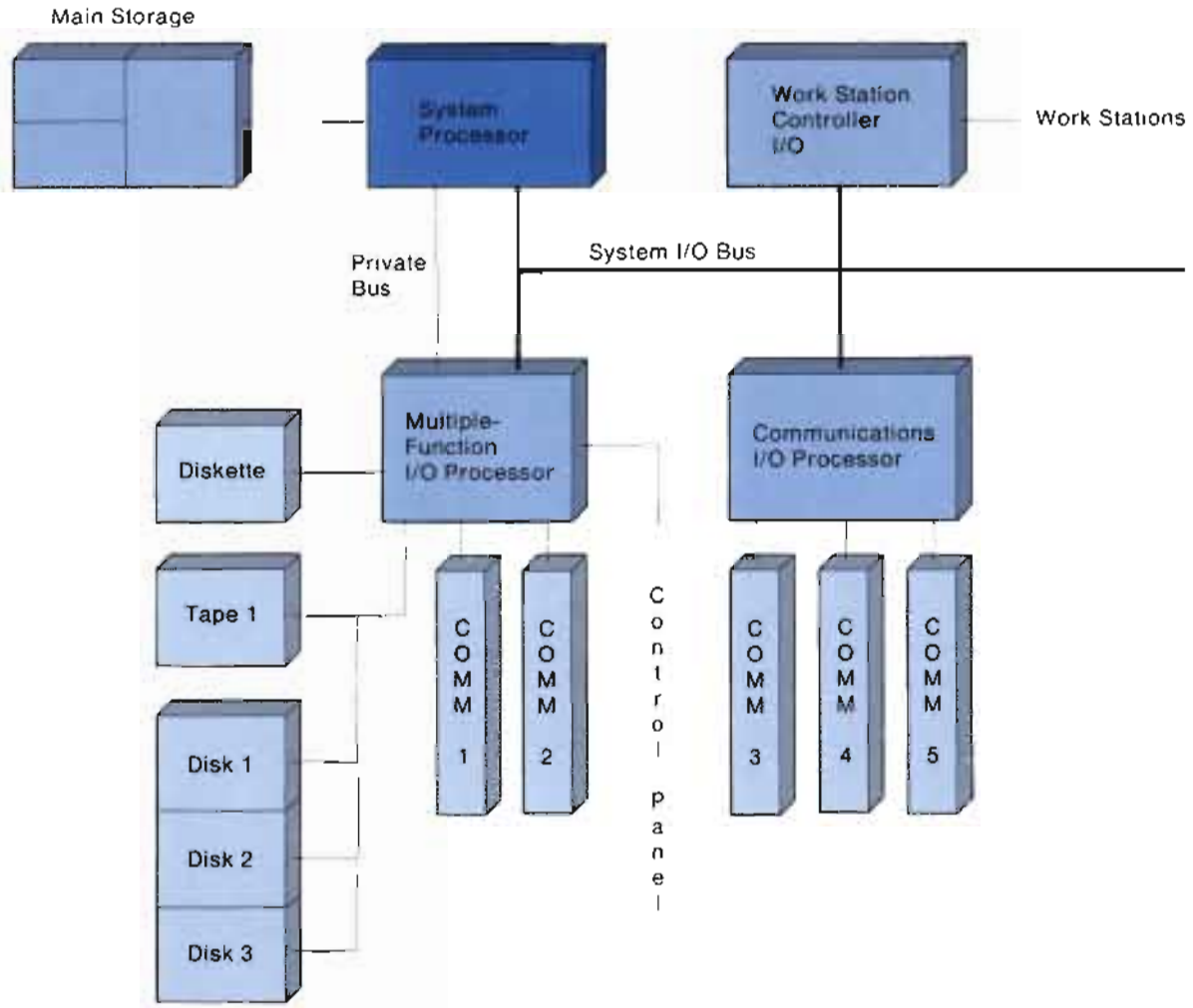


Figure 1 9404 System Data Flow

The Multiple-Function I/O Processor requires all the pieces to be incorporated into a multiple-function design taking one card slot in the system. Two sockets are provided from the Multiple-Function I/O Processor for communications cards. Customers can choose among the three different electrical interfaces for communications, and can mix or match small book communications cards to meet their requirements. This minimizes inventory requirements and lowers the cost for an upgrade; an upgrade involves adding or replacing a communications card, rather than replacing an entire I/O processor card.

The communications I/O processor uses the generic modules (such as the microprocessor, storage controller, storage chips, and the like) to provide a dedicated controller card with three sockets for small book communications cards. Here again, the small book communications cards can be mixed or matched and provide for an inexpensive and efficient method to customize or upgrade.

The magnetic-media I/O processor takes the generic hardware modules and the magnetic-media control chips to provide a one-card, dedicated-function magnetic media controller. This design is packaged on the 9406 System Unit card to provide the larger systems with a standard SCSI bus.

Hardware Technology

The I/O processor hardware is a set of silicon integrated-circuit chips soldered to a printed-circuit card. This card is assembled between covers that form a book assembly, with

connectors on the edges (see Figure 2). Communications cards containing optional hardware in small books can be plugged into these edge connectors (see Figure 3). These small book communications cards customize the I/O processor for particular applications. (For more details on the packaging, see the article *Power, Packaging, and Cooling for the 9404 System Unit*.)

A general-purpose microprocessor with a 32-bit internal architecture and a 16-bit external data bus is used as the programmable controller for the design. The Multiple-Function I/O Processor has 2 megabytes of dynamic random access memory (RAM) for program storage and 64 kilobytes of erasable programmable read-only memory (EPROM) for control of the IPL, diagnostics, and bootstrap loaders.

The microprocessor bus is extended to top-card edge connectors to allow communications cards to be attached. These communications cards are designed to meet the specific requirements for one of three communications electrical interfaces. Any two communications cards can be attached to the Multiple-Function I/O Processor card. (The dedicated communications I/O processor version of the design accommodates any three communications cards simultaneously, and supports much higher aggregate speeds.)

Direct memory access (DMA) data transfers are supported in two modes, single-sided or double-sided. Single-sided DMA transfers occur in or out of I/O processor program storage across the system I/O bus or between I/O processor program storage and an attached I/O device. Thus, relative to the I/O processor, which has an interface bus on two sides, single-sided DMAs transfer data across only one side. Each phase of the single-sided transfers requires processor intervention to set up and start the operation. Double-sided DMA data transfers occur directly between the I/O device and the system I/O bus without being

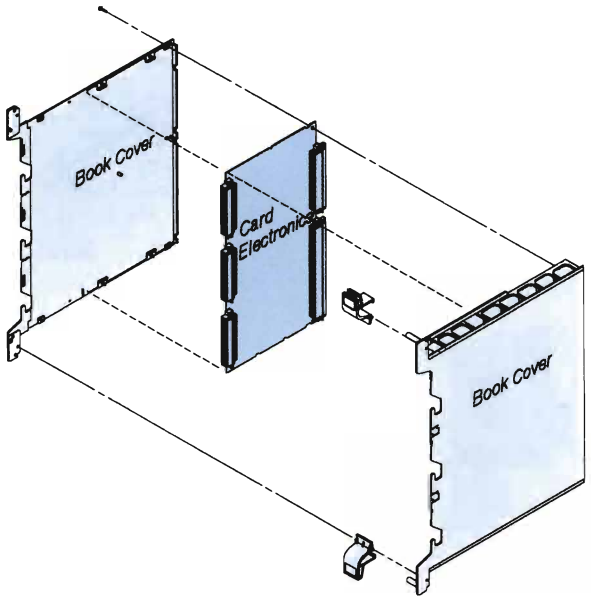


Figure 2 Multiple-Function I/O Processor Book Assembly

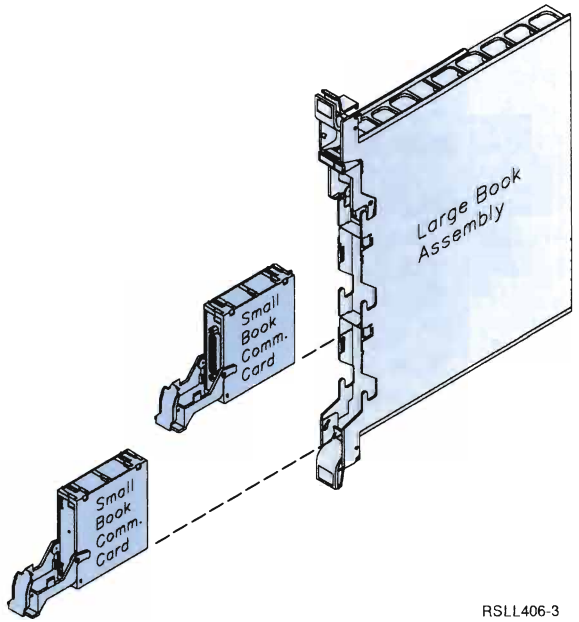


Figure 3 Multiple-Function I/O Processor Communications Cards

routed into the I/O processor program storage. Thus, a double-sided DMA operation transfers data across both interface sides of the I/O processor. Single-sided transfers have the advantage of allowing I/O processor programs to operate on data being transferred. Double-sided DMA has the advantage of speed and reduced utilization of the I/O processor (see Figure 4).

Previous I/O processors supported only two interleaved DMA paths. As one path was processing a transfer, the I/O processor program could be setting up the alternate path. The DMA activity on the alternate path would be started as soon as the DMA completes on the first path. Typically, I/O processor engines spend a lot of time supporting DMA activity because each path is limited to transferring only a single block of data before having to interrupt the processor for a path switch.

In the Multiple-Function I/O Processor design, two significant performance enhancements were added to the DMA data transfer control: **multiple DMA paths** and **minimized processor intervention** through multiple-block transfer.

The first performance improvement, multiple DMA paths, therefore supports seven DMA paths for the SCSI bus, one path for the ANSI 3.8-type diskette interface, and one path shared among the communications cards. Thus, the basic design provides a separate DMA path for each device (the communications lines are lumped together into one device path by this feature). One additional path transfers command and status messages between the System Processor and the I/O processor.

The second enhancement, minimized processor intervention, allows for multiple-block transfers

without the need for block-to-block I/O processor engine intervention. The I/O processor engine sets up the DMA path for the entire transfer, up to 32K bytes. This is achieved by loading the transfer parameters into a series of registers, then setting a status bit in the hardware to indicate the path is waiting to run. When paths that were previously set up finish, the newly set-up transfer will be run in its entirety by the hardware. The processor is interrupted only after all blocks have been transferred.

Any or all of the DMA paths can be set up to perform a DMA transfer across the system I/O bus, but only one is given access at any one time. The DMA paths are granted interleaved access to the system I/O bus based upon a priority established by the hardware. Each path retains access for the duration of a single block transfer. Path priority is re-evaluated after the completion of each block to allow access by paths having higher or equal priority to that of the current path. This implementation provides maximum utilization of the internal bus to support DMA activity with minimum involvement of the I/O processor engine, freeing the engine to perform other processing tasks.

The Multiple-Function I/O Processor SCSI controller provides all the control needed on the SCSI bus to complete an entire read or write sequence to a device without requiring processor intervention. This design requires the I/O processor engine to build the SCSI command block, write it to a buffer, then load the SCSI controller registers with specific DMA transfer parameters. From this point on, the hardware performs all functions needed to send the command to the device, transfer all required data, and receive the ending status. Only then is an interrupt generated to the I/O processor engine indicating completion of the requested operation.

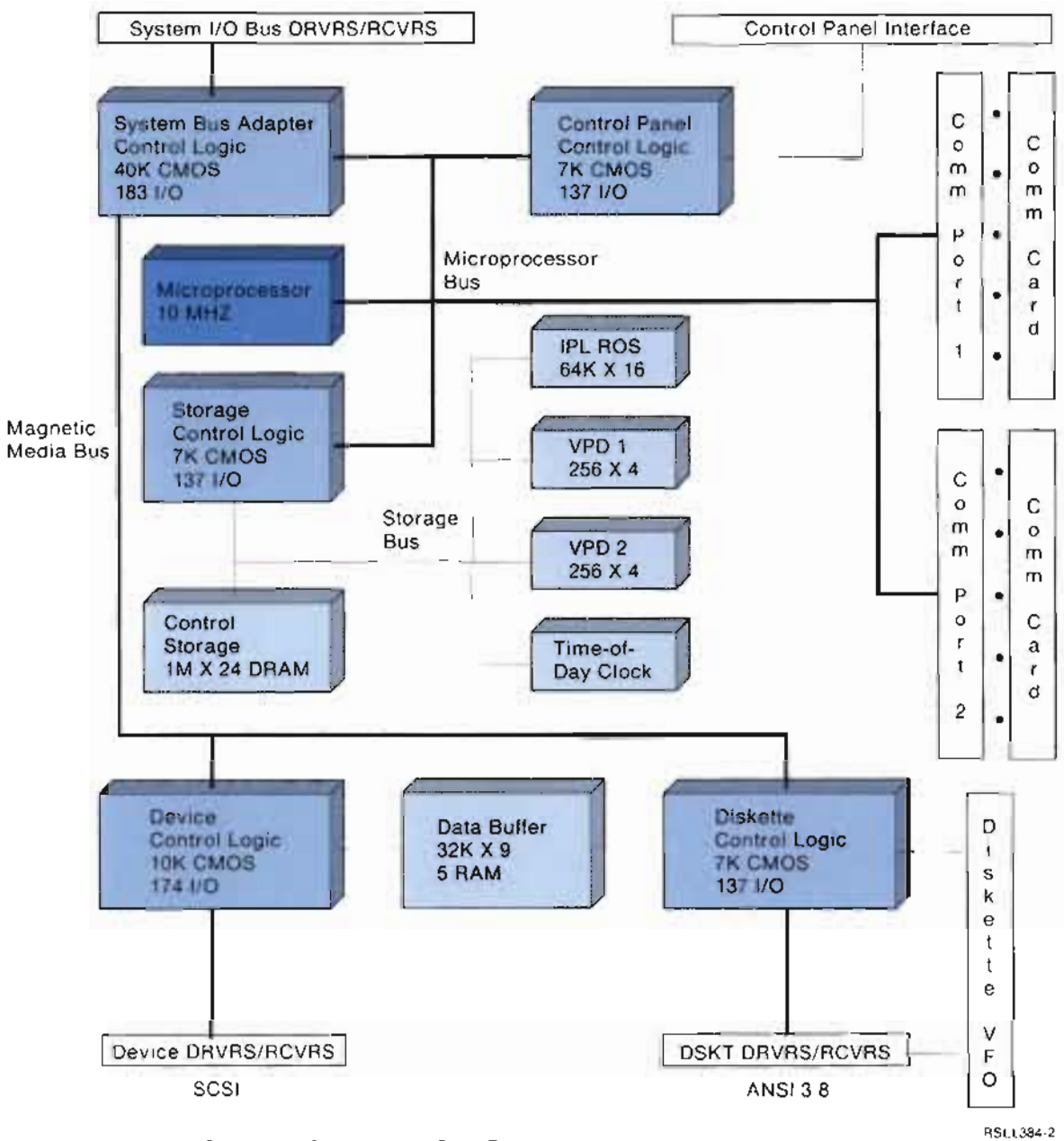


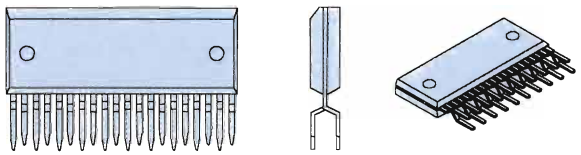
Figure 4 Multiple-Function I/O Processor Data Flow

Electronic Packaging

The functions of each I/O processor are aggressively packaged on single cards, 145mm wide and 280mm high, with: two wiring planes for power on the printed-circuit card; random connections between signal wires on different planes on 2.54mm centers; and up to three signal wires between adjacent connections with standard pin-in-hole component mounting. Maximum utilization of available card space is accomplished by embedding most of the circuitry in custom very large scale integration (VLSI) CMOS chips, and using the new 256K by 4-bit dynamic RAMs available in a ZIP package (see Figure 5).

Four CMOS-I gate arrays are used: one 10,000-cell gate array providing 174 signal pins, and three 7,000-cell gate arrays each providing 137 signal pins. The CMOS-I gate arrays use IBM's 1.5-micron, double-metal process involving nine standard and five personalized mask levels, packaged in a pin-grid-array module measuring 36 mm by 36 mm.

The CMOS-II gate array uses IBM's 1-micron (effective), double-metal process involving 14 personalized mask levels. Re-formed pins on the outer edges of the package allow the 40,000 cells and 183 signal pins to be packaged in a pin-grid-array module measuring 36 mm by 36 mm. Embedded in the CMOS-II array are 6 kilobytes of RAM used for data buffering between the I/O bus and the 10 interleaving data paths internal to the card.



RSLL385-1

Figure 5 ZIP Package

Two megabytes of dynamic RAM, arranged as 1,048,576 by 24 bits, are available for on-card program storage. Twenty-two of these bits provide 16-bit-wide data storage with six bits of error correction circuitry. If a failure is detected in any of these 22 bits during IPL tests, the spare two bits are swapped by the hardware, and the IPL is attempted again. The 24 storage modules use a minimal amount of card space (26 mm by 126 mm), as a benefit of using the ZIP package.

Design Process

The design effort was ambitious. The hardware developers designed 22,000 new circuits, and mapped 13,000 circuits of existing designs into the CMOS technologies. They simulated and built prototypes of the design, verified the implemented functions, and delivered full-function, working hardware to the microcoders on an aggressive schedule.

A single-pass design approach was taken, employing high-level modeling languages to develop new hardware functions. Functions already available but residing in older technologies were converted to CMOS technology using automated mapping tools. Extensive chip-level simulation was used to verify the new functions, as well as the mapped functions that had been merged with the new.

The entire card design was simulated using parallel processor engines to verify the circuitry in a multiple-chip environment. A high-level design language was used to describe the signals between the custom chips and the other card components. Assembly-language instructions for the microprocessor, representing the diagnostic programs that would later be embedded in EPROM, were run against the multiple-chip model to verify the control and data-flow paths between the custom chips, and the microprocessor and its program storage. These diagnostics were again used on the hardware prototypes, to verify the real hardware function. On the 9404, these diagnostics are run every time the card is powered on, to verify continued correct operation.

Reliability

Besides the obvious advantage of lowered cost, the consolidation of the three I/O processor functions into one card controlled by a single processor improves system reliability. This is a result of a reduction in the number of I/O processor engines, their associated program storage and EPROMs, and the bus adapters they require.

Redundancy was used to improve overall reliability of the converged cards. Redundancy is also used in the SCSI data buffer. At IPL, any of the 3K-byte buffer areas reserved for a specific SCSI device can be dynamically re-allocated to any of three back-up areas. The hardware therefore compensates for at least three failures in the SCSI static RAM, or possibly more, depending on the type of failure.

Software Technology

The I/O processor microcode is a set of programs that run in the I/O processor hardware to control the interpretation of commands, the flow of data, and the detection and analysis of possible errors (see Figure 6). This software consists of a set of **common service routines** and a set of I/O control programs, considered as **user tasks**.The Multiple-Function I/O Processor operating system is an object-oriented subsystem. An active object is represented by a running task or process that handles a specific set of work.

The common service routines help to insulate each user task from the specific hardware implementation. These routines are **machine-operating services**, the **Multiple-Function I/O Processor control program**, the **interprocess communications facility**, the **bus transport mechanism**, and the **system bus manager**.

Machine-object services provide four functions: object activation/deactivation, incremental download, object configuration, and subsidiary reliability, availability, and serviceability. The first, object activation/deactivation, allows the I/O processor to have an object loaded in storage only while it is needed. For example, disk objects are always active and thus always in storage; however, certain communications objects are needed only when the System Processor requires those communications services to be active.The second function, incremental download, provides for objects that are not used continuously and therefore do not always need to be in storage (for example, communications protocols). The incremental download function allows these objects, and subfunctions within objects, to be loaded into the I/O processor from the System Processor as they are needed.

Third, object configuration code obtains the resources required by the object (for example, control blocks and data buffers). And finally, additional code provides error logging, performance measurement, and diagnostic testing functions used to isolate problems in the Multiple-Function I/O Processor hardware and software. In Figure 6, this is labeled subsidiary RAS, meaning that it provides reliability, availability, and serviceability for the I/O processor subsidiary of the System Processor.

The next common service code, the Multiple-Function I/O Processor control program, provides a set of operating system services needed to support a multitasking environment. These services include: task-to-task synchronization (using semaphores), message queueing and handling, storage and buffer allocation, initializing and assigning priority to tasks, exception handling, and functions that support interrupts.

The interprocess communications facility provides a means for two processes to communicate with

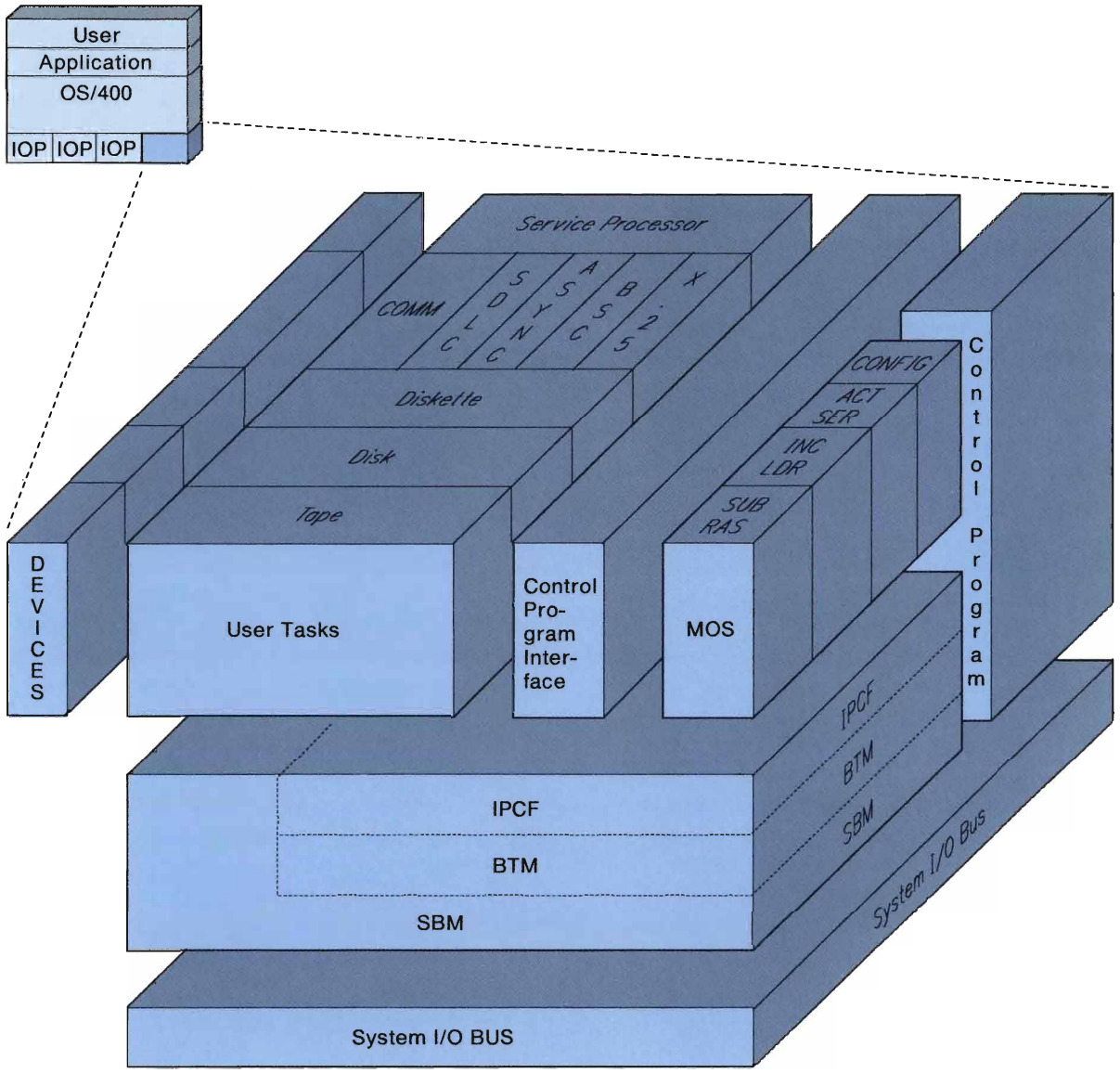


Figure 6 Microcode Structure

each other without either process being concerned with the other's physical location or with the hardware and software used to carry out the communications. The interprocess communications facility is used by Multiple-Function I/O Processor user tasks to open a connection between two tasks (where both are internal to the I/O processor, or where one is internal and one external), receive requests from the System Processor, set up Multiple-Function

I/O Processor hardware to perform the requests, and send responses back to the System Processor.

The bus transport mechanism is used by the interprocess communications facility to move control blocks and data across the system I/O bus (that is, between the System Processor and the I/O processors). It also contains recovery procedures, which are used if errors are detected during the transfer.

The final common service code, the system bus manager, is the microcode interface residing in EPROM that is used to control the actual hardware and service the system I/O bus.

I/O Processor User Task Functions

The user tasks control the three I/O processor functions: **magnetic-media storage device interface, communications protocols**, and **service processor function**.

Three types of magnetic-media storage devices are supported: disk, diskette, and tape units. Although only one copy of task code is in storage, each device attached to the Multiple-Function I/O Processor is provided with a separate device task-control block. This allows other devices to remain operational when any single device detects and reports a hard error. Each magnetic-media storage device task contains the functions needed to: initialize the task; receive requests for the task (through the common service routines); decode the request and translate it into a sequence of device-level commands; perform error recovery procedures for the SCSI bus and storage devices; maintain measurements for the devices; set up the system (using common service routines) and the device hardware; decode device responses; and send formatted responses to the System Processor.

The communications code layers are composed of data-link control, media access control, and port manager layers. These layers, in combination with the operating system code, provide communications microcode for the Multiple-Function I/O Processor card or small book communications cards.

The data-link control layer provides specific protocol support for asynchronous and three synchronous (BSC, SDLC, and X.25) protocols. These four data-link control types form separate tasks, where the machine object services facility loads and activates the protocol dynamically, based on system needs. The code is re-entrant, so multiple lines can share one set of code in the processor. The machine object services code monitors whether the code is being used, and deactivates the code only when all users have finished. The four protocols are fully implemented in these data-link control layers. The media access control layer provides the microcode interface to the small book communications hardware, which provides block check character generation and checking, interrupt generation, a 4-byte buffer, and DMA control. The port manager provides the microcode interface to the communications line electrical interfaces (RS232, X.21, and V.35).

Finally, the service processor user task provides IPL support to start the System Processor, and provides IPL status, system status dump, and problem analysis for the system hardware and microcode. It further provides the interface for the customer control panel, the time-of-day clock, and the vital product data.

Design Process

Hardware simulation is a vital part of the design process which is needed to reduce the development cost, enhance product quality by automating the analysis and verification of the design before prototypes have been built, and speed delivery of a working system. This allows the designers to remove many of the errors normally found after the high-density VLSI chips have been fabricated. This early removal of defects shortens the time needed for all later phases of the debugging process. The specification for each VLSI chip is sent to the manufacturing facility with the idea that they are one-pass-design parts. While some of the chips required minor corrections and thus a second pass, both the overall effort and design cycle time were reduced substantially.

Also crucial to our process is the early development of a microcode simulator. This simulator provides a high-level view of the facilities on the card as well as some system functions, without trying to describe each gate and latch. This microcode simulator was first used to debug the test cases that exercised the hardware simulator. This assisted in the removal of test case errors, which had not been found in manual inspections, and which would have been difficult and tedious to find when searching through the volumes of excruciating detail provided by the full hardware simulator. The key use of the microcode simulator occurred later, when it was used for early debugging of the code that controlled the I/O processor. The I/O processor's control program, and each of the tasks that control an individual device or communications protocol, could be exercised before the hardware returned from the manufacturing facility. As with hardware simulation, the early removal of faults speeds the later phases of the debugging process.

A major trade-off in developing a microcode simulator is the time and resource cost involved to develop a detailed and accurate simulator compared to the benefit of having that improved detail. The simulator did not implement certain complicated features because any errors they helped find could be more effectively discovered and removed in the laboratory. Considerable time was spent designing features to make the simulator easy to use, such as the capability to do full-screen data entry, and displays that grouped registers related to similar functions. Much

debugging of controller code was done on the simulator because it was available early, was easy to use, and was simultaneously available to many people on display stations that were in each designer's office.

The software bringup and functional-unit test process is designed to minimize the dependencies among software components. Rather than require that each software component be debugged and brought up in sequential order, the design process identified a set of functional units that could be brought up in parallel. This significantly reduced the time needed to bring up and test I/O processor functions.

Conclusions

The Multiple-Function I/O Processor provides a subsystem with the full functional combination of three I/O processors and meets the performance requirements of the 9404 System Unit.

Though developed on an aggressive schedule, the Multiple-Function I/O Processor is a high-quality product because a multitasking control-program environment allowed much of the design from three I/O processor subsystems created for the 9406 System Unit to be reused in this design.

The Multiple-Function I/O Processor provides a configuration for the 9404 System Unit with high reliability and lower cost than would be obtained from a combination of single-function cards. Using VLSI circuits and eliminating multiple components like microprocessors, control storage, and bus controllers, contribute to these benefits. Customers are thus provided with an excellent, low-cost system that can be enhanced to gain performance through the addition of dedicated, single-function cards based on the same design.

Power, Packaging, and Cooling for the 9404 System Unit

Reviews the design of the power, packaging, cooling, and acoustics for the 9404 System Unit.

Zanti D. Squillace, Richard A. Tenley, Frank J. Lukes, and Arthur P. Reckinger

Introduction

The AS/400™ 9404 System Unit supports individual removable modules for disk devices, diskette devices, tape devices, a Battery Power Unit, a Feature Power Supply, and feature cards (see Figure 1). The structure has been designed to provide optimum cooling of the components, in addition to protection from radio and radar interference and static electricity. It also ensures low acoustical levels, achieved through optimum fan selection and positioning. For flexibility when choosing components, the devices are packaged so they automatically connect to their respective signal and power connections during insertion. A distributed power system provides a Battery Power Unit, and also provides flexibility for installing future technological enhancements. The modular system structure, the device and logic packaging, the distributed power system, and the Battery Power Unit are designed to make the 9404 System Unit easy to assemble and service, and to allow flexibility to meet future growth requirements.

System Structure

The system structure is a metal unit that houses the system's modular building blocks. The metal chassis was constructed to form an enclosure that protects the system components from outside electromagnetic radiation when they are installed. This was accomplished by controlling the size of air-flow louvres, by specially plating the components, and by using grounding springs on storage devices, the logic cage, and card assemblies. Extensive computer simulation and modeling helped ensure the design would meet or

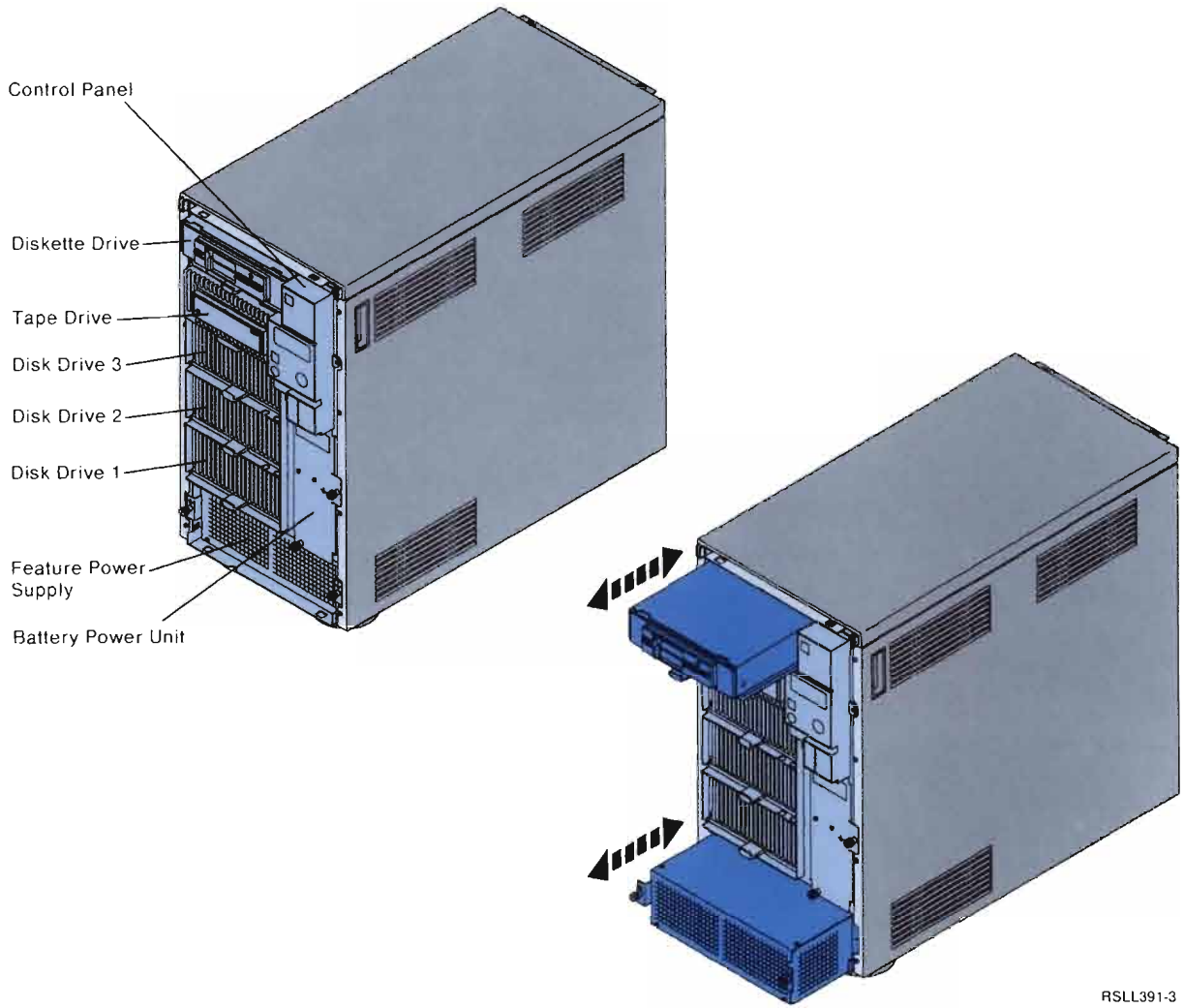


Figure 1 Front Views of 9404 System Unit Showing Multiple Modular Units

exceed all Federal Communications Commission (FCC) regulations.

The system and logic cooling fan is mounted as an integral part of the base power supply. This location yields the best system cooling performance and lowest fan-blade acoustics. Additionally, modular units are used for ease in manufacturing and field service. The tape and disk modular units each contain an additional fan to ensure high reliability.

To align and dock the modular units precisely, the mating connectors were enclosed in a unique, non-conductive floating polymer shell. A three-dimensional computer simulation system was used to ensure reliable docking would occur each time with this design. The resultant docking ability is provided with inexpensive, though highly reliable, industry-standard connectors.

Protection from electromagnetic interference (EMI) and electrostatic discharge (ESD) was a prime requirement in the design of the 9404. The metal chassis is constructed to form an enclosure that protects the system components from outside radiation when they are installed. Flanges on the component assemblies give metal-to-metal contact and, for the storage devices, springs are used in each base to achieve grounding when insertion is complete. EMC treatment of the logic cage, large books, and small books is accomplished using die-cast metal enclosures and grounding tabs at the junctions of each book with the logic cage, and also at the junction of the small book with its large book (see Figure 2).

Early manufacturing involvement was a key item in the design of the 9404. A system that is easy to manufacture was built by carefully designing relatively large subassemblies using common parts, which ensures easy installation of subassemblies, and by maximizing access to all components, fasteners, and cabling.

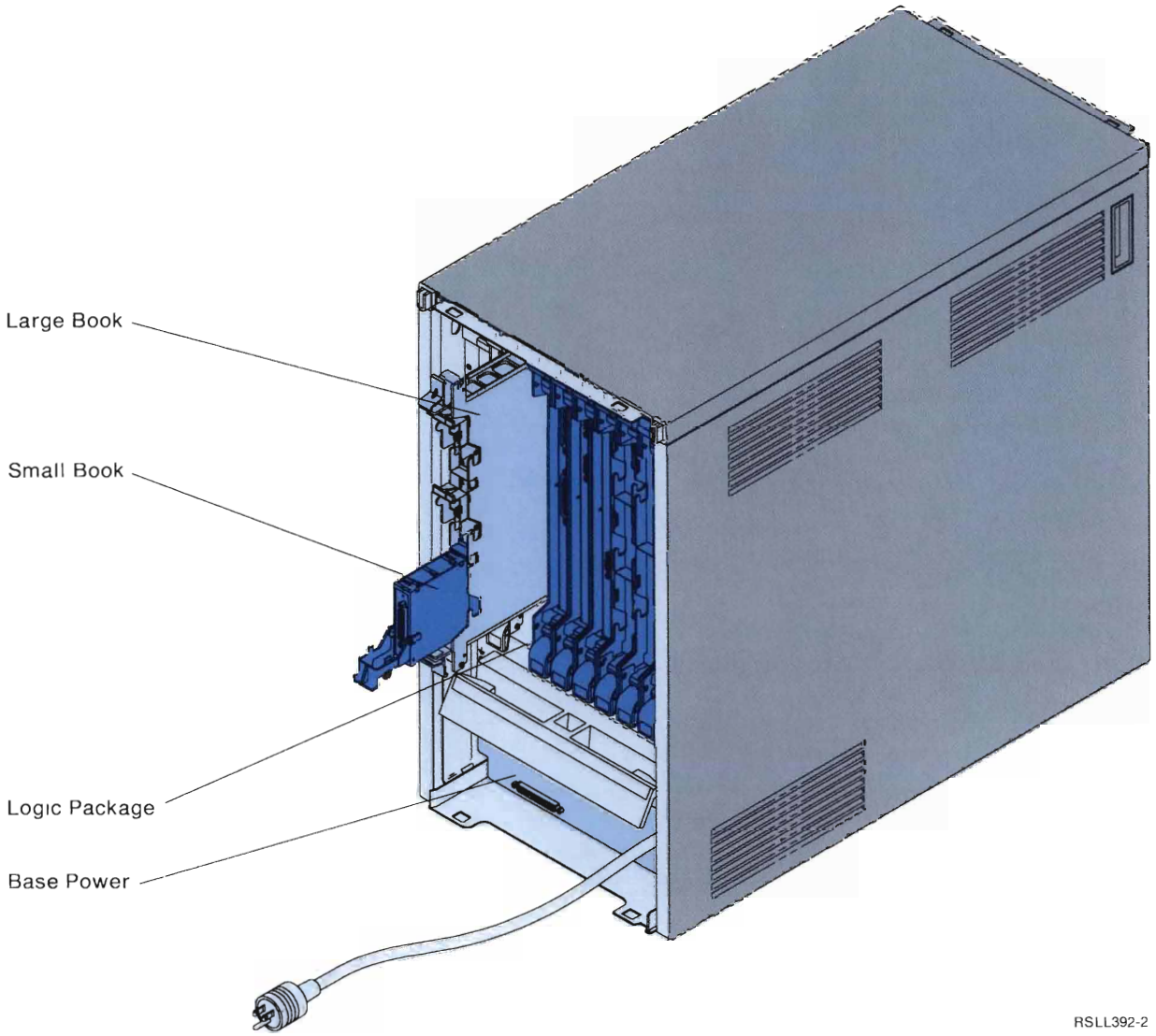


Figure 2 Rear View of System Showing Removal of Books and the Logic and Power Packaging

Designed to be expandable, new devices are added to the system as features. The building blocks provide the flexibility to accommodate new devices developed in the future, including logic families and new storage devices.

Device and Logic Packaging

The disk and diskette modules each contain a power regulator to convert the distributed power (24 volts DC) to the voltages required by the device. In addition, power and signal cables are

provided that connect to the central cabling assembly. This provides flexibility when choosing devices, because they do not have to have the same power requirements and connection arrangement.

The logic-package cage consists of a die-cast aluminum top and bottom, with sheet-aluminum side plates to support the books. This gives excellent light-weight mechanical support, as well as EMC protection. The books are designed with alloy covers for mechanical strength, component protection, tolerance control, and EMC protection. Connections between the logic cards and the backplane are made through industry-standard DIN connectors. These connectors contain a special feature for properly aligning the pins with the sockets in the books.

Distributed Power System

The power system design for the 9404 was changed from the usual multilevel, centralized power system to a distributed power system. The distributed power system was selected over a multilevel centralized power system to provide lower cost and to meet an aggressive design schedule. With the distributed power system, the utility power is converted to + 24 volts DC and routed to regulators located throughout the system. Figure 3 shows the various components of the power system and how they connect. Some of the advantages of the distributed versus central power system are: the power distribution system uses smaller gauge wire, making packaging easier; the loads are next to the regulators, reducing power distribution problems and regulator stability problems, and providing tighter voltage regulation; the regulators are added with the function they support; and the heat dissipation of the regulators is spread throughout the system, rather than just at the power supply.

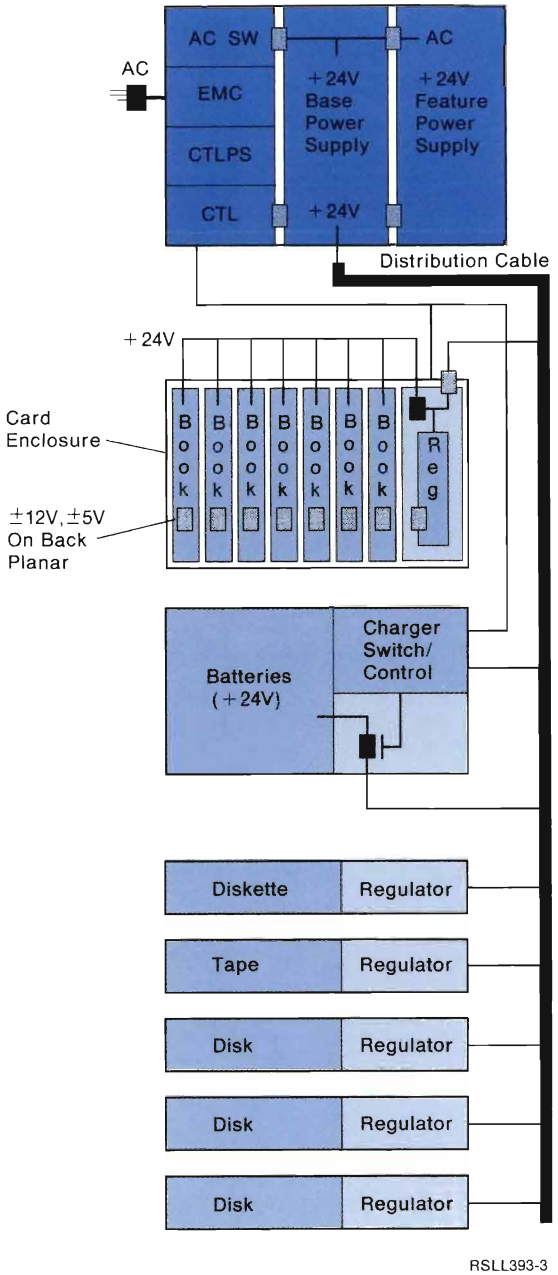


Figure 3 Components of the Power System and Their Interconnection

Battery Power Unit

Distributed power also allowed a Battery Power Unit to be incorporated easily into the power system. This feature provides a function similar to that of an uninterruptible power source. When utility power is interrupted or brownout conditions exist, the Battery Power Unit automatically supplies the power for the system until utility power is restored or until its batteries are exhausted. The Battery Power Unit has a built-in, constant-current battery charger with taper charging to maintain full battery power during normal operating conditions. The charger features over-charge protection and alerts the system when the battery charge falls below half. The Battery Power Unit provides power for at least five minutes on a fully featured machine, which is sufficient to overcome most electrical interruptions.

During the time the system is running using the Battery Power Unit, the entire system continues to function. Many times attached display stations are turned off during the power outage. When their power is restored, users can continue without having to perform an initial program load (IPL) or allow the device to recover before using the system.

Conclusions

The design of the 9404 System Unit offers a modular package that can be upgraded with minimal changes to the mechanical package. This modular package, combined with the packaging, distributed power, and battery power features, provides for ease of manufacturing, assembly, and service. The design is also flexible, allowing for the addition of future improvements and applications.

Improved Methodology for Hardware Quality and Reliability

Describes the unique quality and reliability approach taken to ensure the AS/400 system met its requirements.

Keith L. Thompson and Duane A. Spencer

Introduction

The standard method of determining system quality and reliability has been to compare the complete system target with the accumulated component values as they become available. Such an approach does not focus on individual component quality and reliability early enough in the development cycle to ensure that optimal changes can be made.

An improved method was used to ensure the hardware quality and reliability of the AS/400™ system. It involved establishing quality and reliability targets for each system component in parallel with the functional design.

Approach

Quality and reliability targets were established for all major hardware components of the system (logic cards, packaging, input/output (I/O) units). These targets were based on both capability (technology, function, previous designs) and need (market expectations, anticipated future growth, requirement that new systems exceed replaced systems). Using the component quality and reliability targets, calculations were made to ensure that system quality and reliability requirements would be met.

Component targets did not change based on the system impact of other components. That is, no component was allowed to be of lesser quality because another component was improved. This prevented using the system requirements as a bargaining chip and kept each component on the path of improvement. This method also prevented the system from just marginally meeting its requirements.

Each hardware component was compared to its individual targets using projections of the quality and reliability parameter values of its design and manufacture. These projections were part of product and service cost planning for each hardware component. The component developers were involved in making the projections, as were the system users of the component.

A major advantage of setting targets and making projections early in the design was that key problem areas were identified while improvements could still be made to the design and manufacturing process. This was critical to ensuring the system would still meet its requirements when the designs were completed. The projections also allowed design trade-offs to be evaluated as the design proceeded to achieve the most optimum results.

This approach put quality and reliability on the same level as functionality; it was designed into the product from the start, rather than relying on discovering defects during testing and then making changes to try to meet the requirements. The early quality and reliability design resulted in the use of highly reliable technologies, redundancy and error retry in critical areas, and error correction codes to correct multiple hard and soft main storage errors.

This approach continued with changes to the design, target, and projection values until each major hardware component used by the AS/400 system achieved acceptable quality and reliability parameter values. A flow chart of the approach is shown in Figure 1.

Implementing Quality

Quality involves preventing and removing defects. The hardware design and manufacturing areas set targets for defect-free parameters.

Extensive use was made of design and simulation tools. (See the article *VLSI Design Process for the System Processor*.) Design reviews and formal tests were conducted to measure the defect-removal process compared against projections and targets. The parameters used were the ratio of part numbers released versus part numbers changed and the contribution to the system quality level.

Concurrently, the manufacturing process was structured in such a way to minimize defects while testing to remove those that did occur, and conducting audits to measure the effectiveness of the process. Defect-free rates were projected for the different manufacturing steps and compared to the targets required to meet the system quality-level parameter. (See the article *Manufacturing Card and System Tests*.)

Implementing Reliability

Reliability involves performance over time, and therefore is not as easily measured as defect-free rates. Reliability projections emphasized accurate detail at the hardware-component part level based on the application, testing, and history of the

parts. A file of hardware reliability data was compiled for each major design component to compare with the target parameters for early-life and average-life failure rates. Any component that was over target had to establish a plan to meet the target.

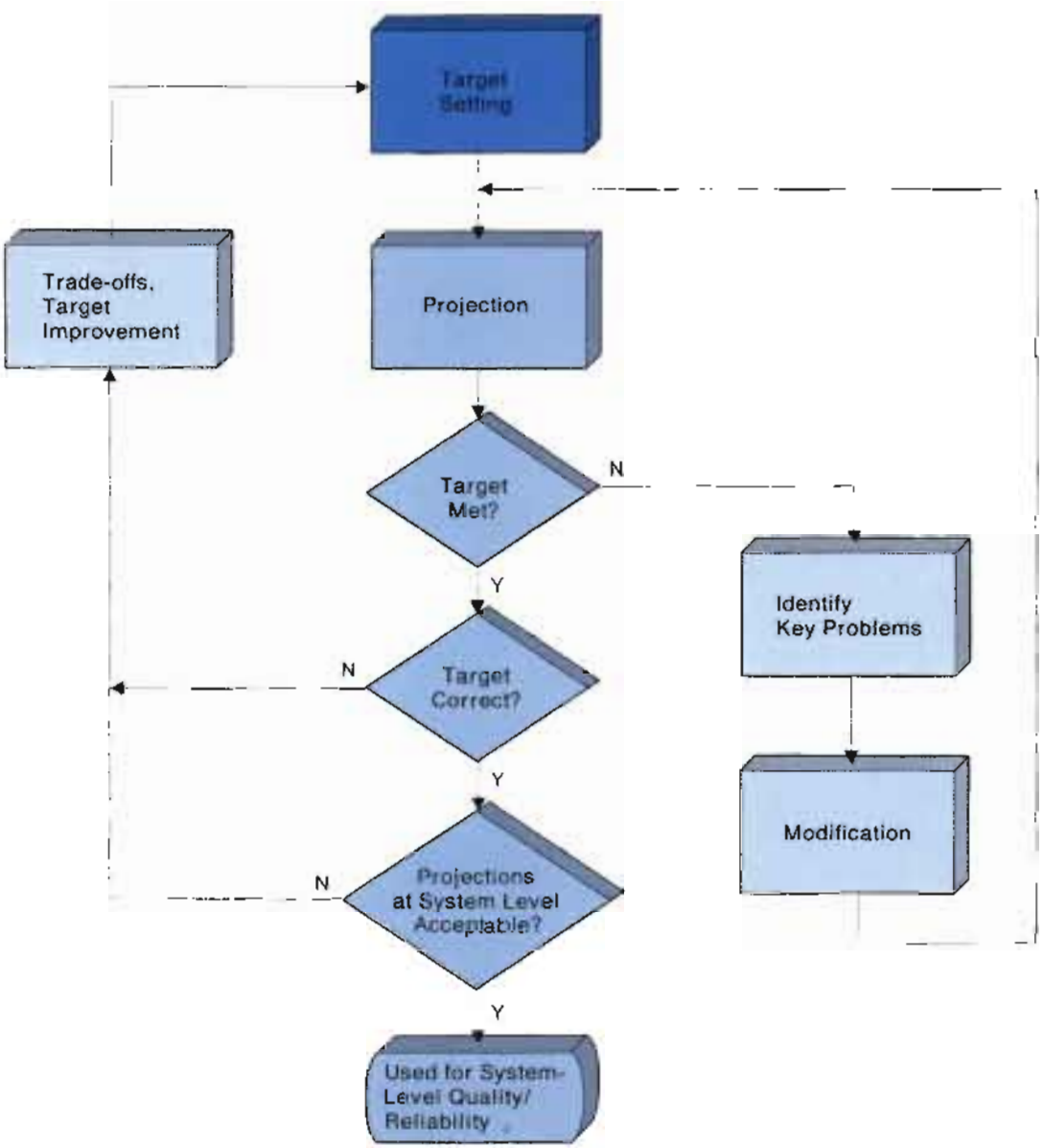


Figure 1 Quality and Reliability Methodology

Results

As projections were made and compared to targets, an iterative set of actions based on identified problems changed the design as it was being completed. This resulted in logic redesign, technology changes, stress screening, manufacturing testing and process changes, as well as target improvements. (The approach is demonstrated in Figure 2.) Each new pass of projections was improved based on information gained from the previous pass and corresponding design changes enhanced quality and reliability.

Among the major changes made to ensure the system requirements were met were: improved cooling to reduce operating temperatures (a 10°C change can increase component failure rates by 50 to 100%), and stress testing during functional operation of components in manufacturing, which removed up to 90% of the residual defects.

Overall, this resulted in a two to 10 times improvement in the quality and reliability values of various components (logic cards, power, and

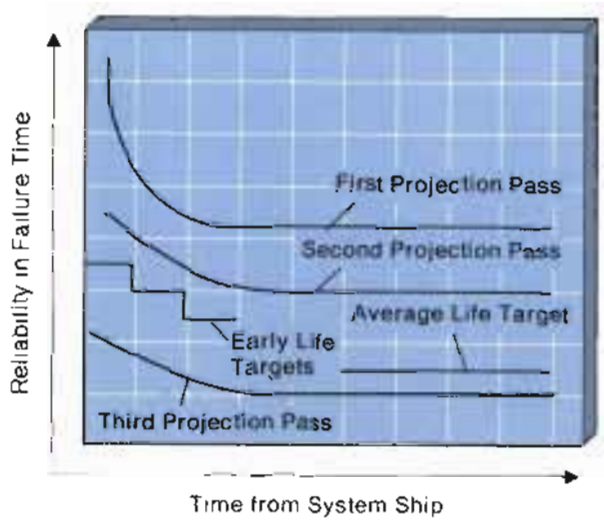


Figure 2 Example of Reliability Improvement

packaging) from the first set of projections to the final one.

Conclusions

Although this method cannot always guarantee that targets will be met, the reasons for unmet targets can be understood and quantified. For the AS/400 system, this approach produced quality and reliability that is superior to that achieved through system-level or post-design modifications alone. The outcome is a system design cycle that increased the average system reliability over four times its initial value, with improvement beyond comparable predecessor products. On the average, the AS/400 logic electronics should not fail during the life of the product, and its magnetic media I/O is the most reliable produced by IBM Rochester, thereby establishing the AS/400 system as the new standard for quality and reliability.

Acknowledgements

The authors wish to thank the reliability and serviceability work group and the reliability department for their work in establishing the basis and tools for this approach.

Design of the IBM 9332 Disk Unit

Presents the hardware improvements and changes in design philosophy and procedures that provide the high capacity per cost, performance, and reliability of the IBM 9332 Disk Unit.

Earl A. Cunningham

Introduction

The IBM 9332 Disk Unit has 200-megabyte and 400-megabyte versions. A photograph of an open 400M file used in the rack-mounted 9406 System Unit is shown in Figure 1.

The 9332 Disk Unit incorporates significant improvements in capacity, performance, cost, and reliability. This not only includes advances in the basic magnetic components, but also significant changes in design philosophy. These, together with advanced electronics and data handling processes, allow improvements in capacity and reliability significantly above that expected from the improved magnetic components alone.

The design philosophy used for the 9332 Disk Unit emphasized the basic benefits of minimum cost for capacity, high performance, and high reliability. The capacity per cost is increased by: increasing the area used for data, using that area more efficiently, recording at the higher density achieved by better components and improved data processing, and reducing production costs. The Disk Unit has improved seek times and a higher data transfer rate. The reliability of the Disk Unit is increased by basic improvements in the head and disk, and manufacturing quality control. The reliability and capacity are also improved by additional recovery procedures for failures that might occur.

These improvements are addressed in three categories: the physical file design, electronic design, and data handling. (For additional

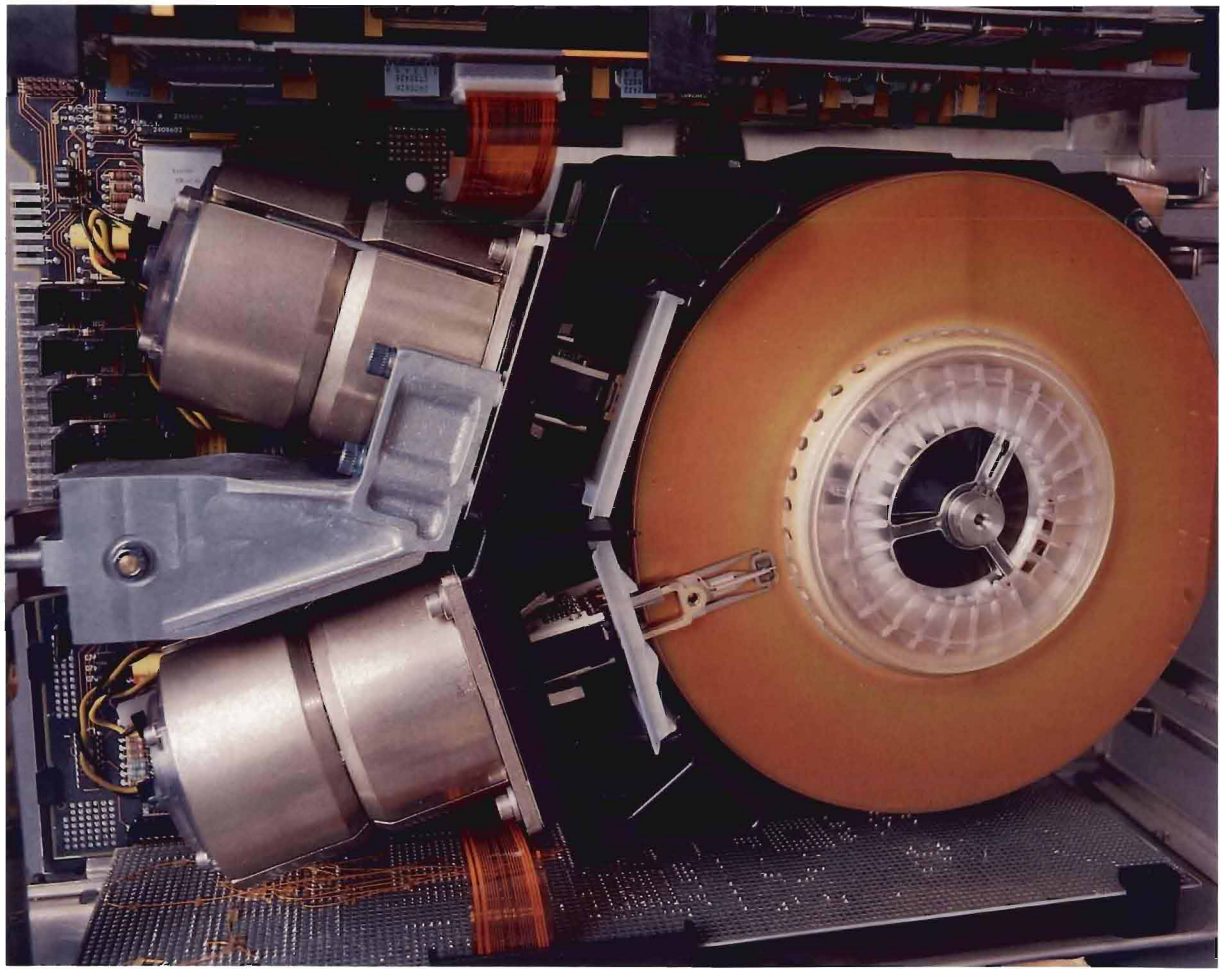


Figure 1 An IBM 9332, 400M Disk Unit with the Covers Removed

information, see the article *The Disk Manufacturing Process*.)

Physical File Design

Physical improvements were made to the heads, disks, and actuator. The head is a manganese-zinc monolithic of IBM, Rochester, MN, design. The design improves the head efficiency, which improves the signal-to-noise ratio compared to that obtained with standard monolithic heads [1]. The disk's particulate coating is prepared using a new coating technology [2, 3] that provides a much smoother surface, better particle dispersion, and fewer and smaller defects than those obtained with standard processes. The actuator system for the 9332 Disk Unit (400M) has two separate actuators (see Figure 1) for faster operation and more operations per second.

The usable disk area of the 9332 is significantly increased from previous products, providing more useful recording space. Area usage is improved by combining the sector-identification field with the data field, so that only one (rather than two) synchronization field is necessary [4]. This improves the format efficiency, providing additional space for data, and improves reliability by including the identification field within the data error-correction algorithm. Because both the identification field and data are written simultaneously, the chance of them being misaligned is eliminated.

Another small factor in providing more surface for data is the allocation of more alternative sectors at the inner tracks, where the probability of defects is higher, and fewer alternatives at the outer tracks [5].

The mechanical integrity of the head to disk interface is another important aspect of the physical file. The development of a reliable head to disk interface is a difficult problem, involving the characteristics of the head and disk material, the physical flatness of the head and disk, the fly height for both steady-state and dynamic-access conditions, and the effects of environmental variations. Other concerns include possible disk-clamping distortions, the quantity and movement of the added disk-lubrication material under various types of operation, and many other variables. A significant number of personnel and amount of equipment was dedicated to the in-depth investigation of these effects, resulting in the selection of the head to disk fly height for the best possible signal-to-noise ratio while providing a very reliable design. This was accomplished at a small cost per file due to the large number of Disk Units being built.

The mechanical integrity is also maintained by the disk enclosure design and manufacturing techniques that minimize contamination. One of the most significant contaminants is magnetic particles from permanent magnets that find their way to a head. If they attach to a head near the disk surface, some demagnetization of the medium can occur, thus degrading the signal-to-noise ratio of the recording. While the magnets in a disk unit are normally coated to prevent magnetic particle loss, the 9332 goes a step further. The disk enclosure is closed before the actuator and motor magnets are installed. These components are outside of the disk enclosure. This allows the heads and disks to be assembled into the 9332 in a magnetically clean area, and the inside of the completed disk enclosure thus remains magnetically clean for the life of the 9332.

Electronic Design

The electronic design of the 9332 provides many advantages. One is the use of an improved baseband recording code, a run-length limited RLL(1,7) code with a two-thirds rate [6]. The 1 refers to the minimum number of consecutive encoded zeroes and the 7 to the maximum number of consecutive encoded zeroes between encoded 1's. The recording code most often used for current disk units is a run-length limited RLL(2,7) half-rate code. For the same data rate, the RLL(1,7) two-thirds rate code has a maximum recorded-transition density 12.5% higher than that of the RLL(2,7) code, which causes the RLL(1,7) code to have somewhat more bit shift than the RLL(2,7) code. However, because the two-thirds rate code has one third more time for each encoded bit to be detected than with a half-rate code, a significantly larger tolerance for bit shift is allowed. The RLL(1,7) code used in the 9332 thus provides about 5% higher capacity than that obtained with the RLL(2,7) code.

Another significant improvement is in the arm electronics module, which is a head-signal preamplifier. Standard preamplifiers have a damping resistor across the input to damp the head resonance during write operations. However, during reads, the thermal noise this resistor generates significantly contributes to the total electronic noise. The new amplifier has a network that damps the write-current waveform without adding any extra thermal noise during reads, thus improving the signal-to-noise ratio. Without the damping resistor during readback, the head has an under-damped resonance. This increases the high frequency data signals nearer the resonance, which outweighs the noise increase due to the higher source impedance. This further improves the signal-to-noise ratio and also provides some of the required equalization of the readback signal. The increased signal-to-noise ratio with the new design allows the recording density to be increased about 12%.

Another improvement is the use of a single-burst error correction code (ECC) as a first recovery procedure. If a single-burst error occurs, the data from the next sector is read and pipelined while the correction is being made to the first sector.

The data buffer and fast-parallel interface normally allow the file to continue reading without time lost to added disk revolutions. Only sectors with two or more error bursts require additional revolutions to reread the data. This ECC allows the recording density to be increased about 10% with the resulting higher soft-error rate compensated by the ECC. The measured performance of the 9332 shows that typically over 99% of the soft errors are corrected by the single-burst ECC.

Another improvement is the fault-tolerant synchronization byte for each sector. The tolerance allows the proper starting point of each sector to be identified, even when a soft error occurs in that byte [7]. This feature reduces the number of missing-sector failures.

The addition of microprocessor control provides an improvement by allowing optimization of each head and disk combination. During 9332 assembly, one of eight different write-current values and one of eight detection parameter (delta-v) values for each head at each of three radial bands may be selected to optimize the performance. These values are stored on the 9332 and are loaded into active storage for each power up. The improved performance can be exchanged for a 5% increase in recording density.

The new servo code and control system used in the 9332 provides an improvement by allowing the other described improvements their full capability. The new servo system provides the smaller head-to-track misregistration required at the higher track densities, where the smaller track width causes more rapid signal loss with off-track distance. The higher linear density reduces the side fringing of the recorded track, which saves space but does not separate the data as far from interfering signals, further restricting the tolerance for off-track distance. For more information, see the article *Digital Servo Control for Disk Units* [8, 9, and 10].

Data Handling

The self-testing capabilities of the 9332 assist in determining the optimal current and detection parameters (delta-v) for each head. The 9332 performs its own surface-analysis testing, locating any defective sectors. These reduce testing costs by eliminating file testers. In addition, several data processing procedures are used in the data recovery procedures and in data quality maintenance, improving performance and reliability and allowing further capacity increases.

Previously, today's recoverable soft errors were unreadable hard errors. Thus the early error rates were required to be nearly as small as the hard-error rate. With a modern system, most errors are due to electronic noise and not due to any actual flaw in the system. Maintaining a very low soft-error rate required lower recorded densities to obtain the higher signal-to-noise ratio, which, by today's standards, represents a poor use of the available signal. Recording at higher linear and track densities results in more capacity and a lower signal-to-noise ratio, with a corresponding higher soft-error rate. The higher rate can easily be handled by the 9332's data recovery procedures. The higher soft-error rates are associated with the natural electronic noise deviations that are uncorrelated for each reread, which allows very effective recovery. Although not immediately obvious, the design using higher density with the higher error rate actually improves performance over low error-rate designs.

In the 9332, the areal data density was increased by 20% by allowing the corresponding increase in soft-error rate. (For best results, improvements are taken partly by increased track density, and partly by increased linear density.) During typical operation, the recovery time from the increased soft error rate reduces throughput by less than 0.1%. However, the increased data rate with the higher linear density actually increases the throughput by a much more significant amount than the minor loss due to the recovery times. Thus, allowing an increased soft-error rate not only allows increased capacity, but also increased throughput.

The 9332 has extensive data recovery procedures that are rapidly accomplished with the internal control. In addition to the initial single-burst ECC, the recovery procedures include the standard methods of rereads on track and off track, but combined with the ECC. Where a normal recovery procedure would attempt a single-burst ECC, recovery, the 9332 provides a double-burst correction capability for data recovery. This allows the correction of two independent errors within the sector.

After all the normal data recovery procedures are done, the remaining errors would normally be hard errors. However, in the 9332, a unique function has been added to the recovery procedure, which virtually eliminates the dominant cause of hard errors. This condition can occur if the adjacent tracks overwrite part of the track of interest to such a degree that no head position allows the data to be read correctly. Because the data tracks are placed close together to obtain high capacity, this large interference can occur with low probability and under extreme conditions for heads near the maximum of the track-width tolerance distribution. Figure 2 shows a read-head gap, with partial edge sensitivities indicated by the dark edge area. The head is over a data track, T2, with severe interference from adjacent tracks T1 and T3 due to track misregistration, with the written positions both inward from their normal position.

Previously, to keep the probability of a hard-error occurence to less than once in the life of the disk unit, the written tracks had to be sufficiently separated so that only deviations greater than 5.5 sigma in track misregistration would cause the

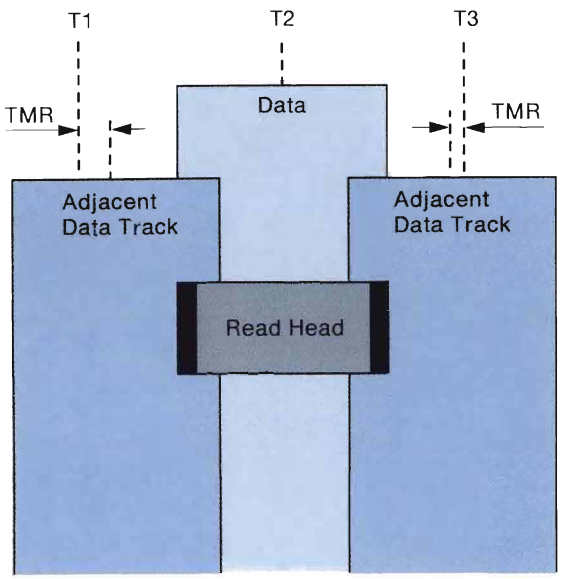


Figure 2 Read-Head Gap Over Data Track with Severe Interference

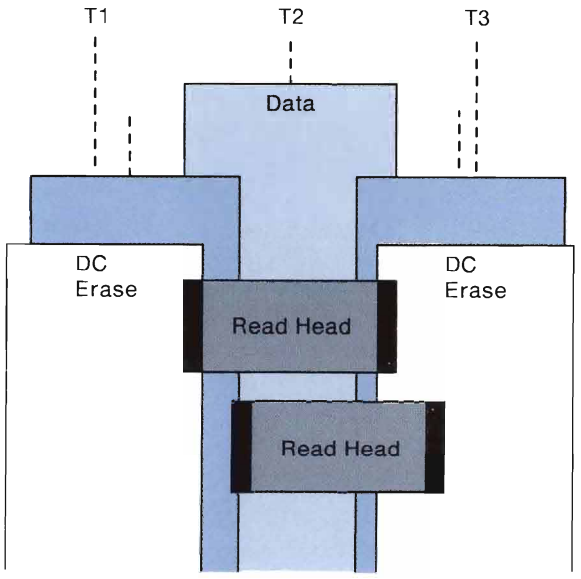


Figure 3 Recordings with the Interfering Tracks Erased

failure. For such failures, it was found that the remaining signal from the partially overwritten track was more than enough to recover the data correctly if the interfering signals could be excluded. The 9332 was thus designed to read each adjacent track sector, store and verify the information, and then DC-erase the adjacent sectors [11]. Figure 3 shows the interfering tracks erased. Because the track misregistration is almost always less than the extreme value that caused the severe problem, erasing tracks T1 and T3 does not erase all of the interfering recorded signal, so the sector on track T2 may still not be read correctly. If this occurs, the head is offset to find the position that avoids the stronger residual interfering signal (as indicated in the figure), where correct reading of the data is much more likely.

If recovery is still not possible with the head on track or in multiple off-track positions, the adjacent sectors are again DC-erased with the head offset inward toward T2. Reading is again attempted at several positions. If this is not successful, the adjacent sectors are erased again with more inward offset, and reading is again attempted at several off-track positions. Because the exact position of the remaining recorded track is not known, gradual erasing guarantees that, when sufficient interfering signal is removed, the largest possible data signal strength remains. After recovery, the data and the adjacent sectors are rewritten and verified for accuracy. This process is so effective in recovering data from extreme interference situations, the disk unit can be designed with the tracks placed closer together without causing a data loss due to track-to-track interference. The increased 9332 track density provides 4% more capacity, improved protection against interference failure, and minimal error-recovery impact on throughput. The cost of the added capacity and protection gained by this procedure is small. The microcode program required for this procedure is stored on the 9332 and is read and loaded into the microprocessor if it should ever be needed.

Another unique process added to the 9332 is that of data-quality maintenance [12]. This procedure provides protection from various small degradations from continued track-misregistration effects, slight changes due to aging of components, or other small problems. While occasional rereads are required due to the normal extremes of electronic noise, more than a few steps of the data recovery procedure indicate a possible degradation. In this case, the sector number and a score related to the depth of recovery are entered into a large raw-data error log. If a previous score exists for the sector, the new score is added to the previous total. A sufficiently high score indicates a degraded signal, so the sector is rewritten and verified for accuracy. It is then listed in a smaller log of rewritten sectors and the raw-data error score is reset to zero. Multiple small scores, as well as one high score, will cause a sector to be rewritten. The verifying procedure uses a reduced recovery procedure, because the data should then be of good quality. Data not verifiable with simple recovery procedures indicates a magnetic defect in the sector and the sector is recommended for reallocation. With proper data verification, no recommendation is issued. The microprocessor uses disk unit idle time to analyze the error logs. If this shows that significant errors have reappeared on a previously rewritten sector, a recommendation for reallocating the sector is made. If either log is full, the newest entry replaces the oldest entry. At higher error rates, the log's data is flushed out faster, so that only sectors with error rates significantly above the disk unit average are identified.

Using this process, small degradations can be eliminated by rewriting weak sectors. The process

allows for different error rates in different disk units, which prevents a high occurrence of rewrites for high, but normal, error rates. This process gradually eliminates the weaker sectors of a disk unit during its early life, assuring a more uniform and more reliable disk unit. This process automatically occurs for any sectors being read. For sectors that may have no read operations during normal customer use, protection is provided when a periodic backup reads all of the disk unit's data. This process thus refreshes data and cleans out the poorest sectors on the disk unit, providing continued data protection over the disk unit's life.

Other logging and analysis of non-data errors is also done in the 9332, providing early detection and analysis of possible electronic degradations. This provides additional safety for the stored data.

Development

Most of the design aspects of the file are tested on several precision test stands, such as the one shown in Figure 4.

The test stands include air-bearing spindles and laser-controlled access slides for precision head and disk testing in an enclosed clean-air chamber. The operation of the spindle, access slide, data channel, and data handling and analysis hardware are exercised by computer-controlled test programs for analyzing the error rate.

The effect of data-recovery algorithms is also tested on a precision test stand. The results provide better understanding of the recording system, which supports new developments. The tests also quantify the value of each new development. Additional testing of each new development is done during disk unit operation to verify the benefit expected.

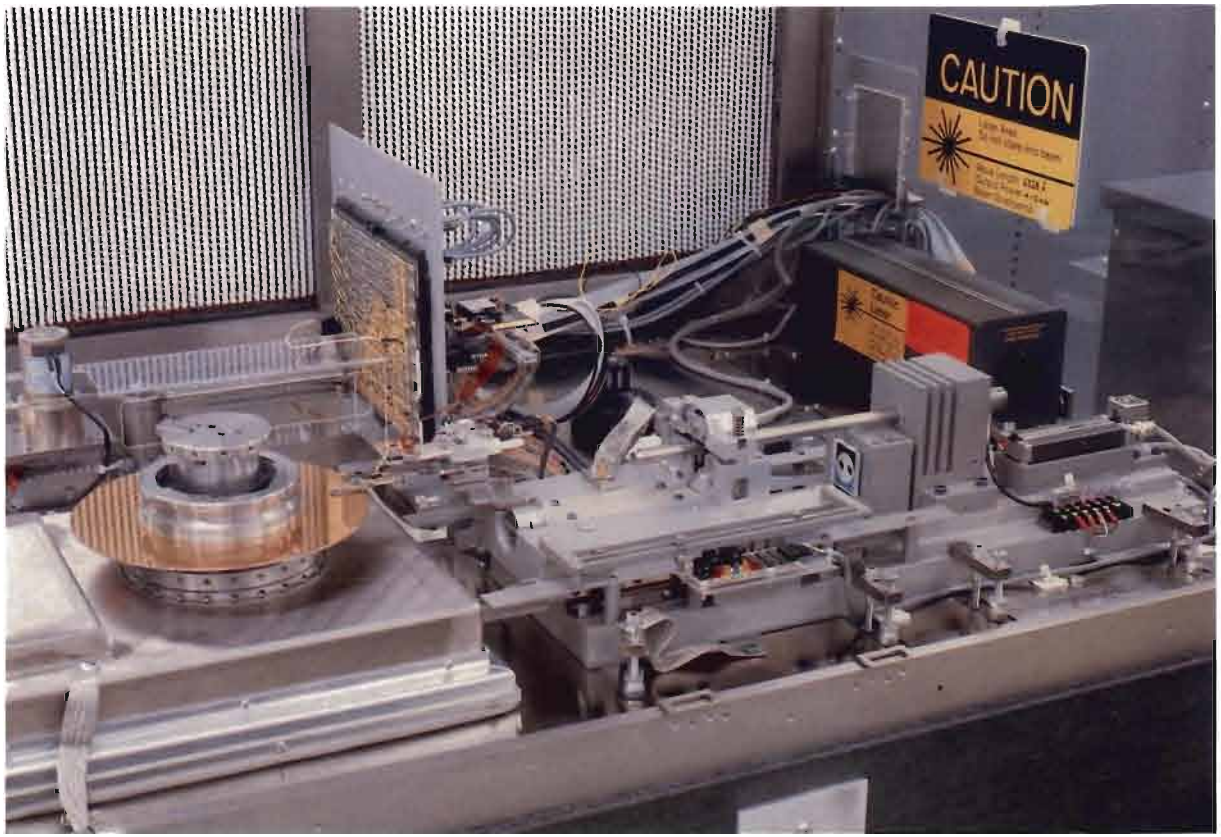


Figure 4 A Precision Test Stand Used in Disk Unit Development

Conclusions

The IBM 9332 Disk Unit includes basic head and disk improvements that increase the available signal level. A new data preamplifier design reduces the electronic noise level. The new design philosophy allows better use of the signal-to-noise ratio, resulting in more capacity. Single-burst error correction without loss of disk revolution time and self-implemented data recovery procedures permit a higher recording density and increase the throughput. A patented recovery procedure and patented servo system allow tighter packing of data tracks, again providing more capacity. A data-quality maintenance system prevents long-term degradations of the data, and the self-testing capability of the 9332 reduces cost. These improvements provide better capacity per cost, performance, and reliability. Future disk units will continue to benefit from many of the improvements first developed for the IBM 9332 Disk Unit.

References

1. LeVan, D. (IBM Rochester), *Reverse Window Ferrite Head*, **IBM Technical Disclosure Bulletin,** Volume 26, Number 3B. August, 1983.
2. Peugh, H.V. and A.W. Ward (IBM San Jose), *Coating Thickness and Wedge Geometry Control for Magnetic Disks*, U.S. Patent 4,485,758.
3. Hagen, J.A., M.A. Wilke, and J.E. Maloy (IBM Rochester), *Magnetic Disk Coating Method and Apparatus*, U.S. Patent 4,587,139.
4. Greenberg, R. and D.A. Styczinski (IBM Rochester), *Sector Identification Method for Hard Sectored Hard Files*, U.S. Patent 4,656,532.
5. Anonymous (E.A. Cunningham, IBM Rochester), *A New File Defect Strategy*, Research Disclosure, Number 267 July, 1986.
6. Adler, R.L., M. Hassner, and J.T. Moussouris (IBM Research), *Method and Apparatus for Generating a Noiseless Sliding Block Code for a (1,7) Channel With Rate 2/3*, U.S. Patent 4,413,251.
7. Cunningham, E.A. (IBM Rochester), *A Fault Tolerant Sync Byte for RLL(2,7) and RLL(1,7) Codes*, **IBM Technical Disclosure Bulletin,** Volume 29, Number 1. June, 1986.
8. Collins, D.W. and F.E. Axmear (IBM Rochester), *Phase Modulated Servo System*, U.S. Patent 4,549,232.
9. Collins, D.W. and M.A. Weed (IBM Rochester), *Phase Difference Demodulator*, U.S. Patent 4,642,562.
10. Ottesen, H.H. et al (IBM Rochester), *Adaptive Control Technique for a Dynamic System*, U.S. Patent 4,687,127.
11. Cunningham, E.A. and D.C. Palmer (IBM Rochester), *Error Recovery Procedure Using Selective Erasure,* U.S. Patent 4,516,165.
12. Cunningham, E.A., D.W. Hegeman, and D.A. Styczinski (IBM Rochester), *Prevention of Hard Errors in Magnetic Files Due to Long Term Degradation*, **IBM Technical Disclosure Bulletin,** Volume 92, Number 10. March, 1987.

Digital Servo Control For Disk Units

Describes the microprocessor-based head-positioning servo control for the dual actuators in the IBM 9332 Disk Unit.

Hjalmar H. Ottesen

Introduction

With the availability of low-cost microprocessors, a revolution is under way in the design of servo control for electromechanical systems. Typically, the servo control was designed using amplifiers, resistors, capacitors, inductors, and continuous sensor inputs. This type of control, called analog servo control, has been used to position recording heads in disk units for more than two decades [1]. The discrete components in an analog servo control system have initial tolerances that will drift with age and temperature. This may cause less than optimal system performance.

Another type of servo control is digital servo control, in which all the resistors, capacitors, inductors, and amplifiers are physically replaced with a microprocessor. The sensor inputs are sampled and the measurements are converted from analog-to-digital (A/D) representation so that the inputs can be understood by the microprocessor. In the same fashion, the output control signals from the microprocessor must be converted from digital-to-analog (D/A) representation to actuate the electromechanical system. In a digital servo control, all the values of the system parameters are represented by digital numbers, which do not change with age or temperature unless programmed to do so [2].

A full-scale move from analog servo control to the more reliable, less costly, and higher performance digital servo control has begun. The IBM 9332 Disk Unit is part of this movement. It has a 400-megabyte capacity on four disks, with average access time less than 20 milliseconds. The 9332 is the first IBM disk unit to have recording-head positioning under total control of a microprocessor, giving it the best tracking performance of any announced IBM storage device.

Digital Servo Control

The objective with a high-performance storage-device actuator's servo control is to move the recording head from one track to another, and to settle on that track within a small off-track error limit in the shortest possible time. After the head has settled on the track, the servo must be able to keep the head on the track within the specified off-track error limits.

Figure 1 shows a conceptual block diagram of the 9332 digital servo control. The recording head, reading magnetically pre-recorded patterns forming 74 radial servo sectors, produces position-error information. These radial sectors are interlaced with 74 radial data sectors on each of the eight disk-recording surfaces. The analog position error signal, indicating the amount the recording head is off track, is converted to digital by an A/D and fed into the microprocessor every 260 microseconds. The microprocessor has stored microcode representing the estimator and controller algorithms, and will output a digital control signal for each sample period. A D/A conversion performed on the control signal results in an analog signal that is amplified by the current driver to drive the actuator voice-coil motor, keeping the recording head on track.

The model of the actuator is a sampled double integrator, with bias forces and time delay. The microprocessor stores track position, estimated velocity, estimated bias force acting on the actuator assembly, and integrated track-position error, which are all processed every sampling period. These variables are called **state variables** and are modeled as discrete-time linear difference

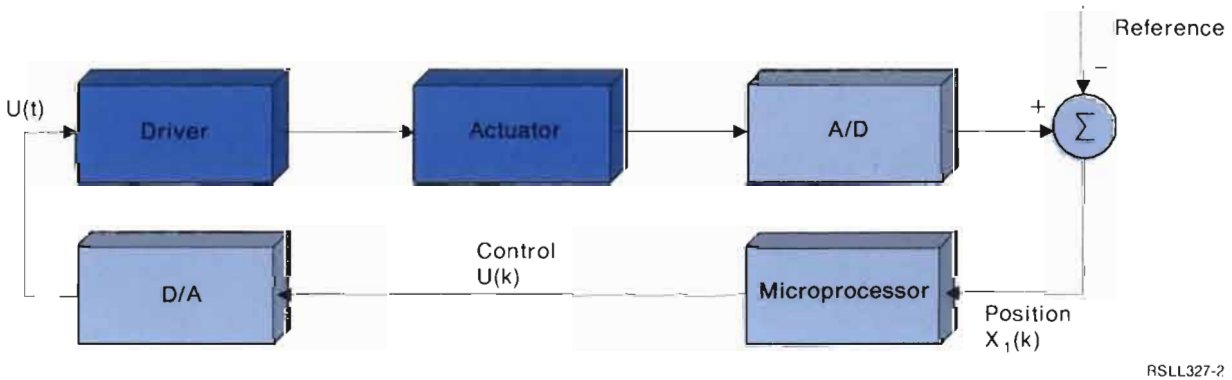


Figure 1 Concepts of Digital Servo Control

equations. (Mathematical expressions for the discrete-time dynamic actuator model are shown in Figure 4, equations 1 through 5; for additional information, see [3]).

Estimator Description

In any high-precision servo design, sensor inputs from the electromechanical system for states of position, velocity, and bias are required. In general, the more relevant, independent, and noise-free the sensor inputs, the better the overall servo performance. Sensors, however, are generally expensive and add complexity to the design. The microprocessor, containing an approximate dynamic model of the electromechanical system, can, with just one or a few sensor inputs, compute other states (for example, velocity and bias). The estimated states can now be used in the controller-feedback algorithm, together with the measured sensor states, to yield good overall performance at low cost.

With each position-error signal measurement, a corresponding control signal corrects the head position. The new estimated velocity and bias are calculated from the present position-error measurement and previous values of estimated velocity, bias, and controller output. The estimator design is independent from the controller design. Its dynamic response for this disk unit was determined by two estimator gain constants. The estimated velocity is used as one of the inputs to the controller, and the estimated bias is only used internally for removal of bias errors from the velocity estimates. This state, called previous control, is used as another variable in the estimator to eliminate the effects of delay. (The digital estimator algorithm is shown in Figure 4, equations 6, 7, and 8.)

Controller Descriptions

For each sampling period, the controller outputs a linear combination of measured and computed states. The controller has two different modes: the track-follow mode and the track-seek mode.

Track-Follow Mode

The purpose of the track-follow mode is to keep the recording head on track in the presence of actuator disturbances such as bias forces, external vibrations, disk spindle bearing run-out and imbalance, and noise in the position measurement. The four state variables (position, estimated velocity, integrated position, and previous control) are each being multiplied by separate control-gain constants. The sum of these four products is fed back and is the new updated control based on the current position-error measurement. (See Figure 4, equation 9, for the mathematical expression of the track-follow algorithm.) The controller output is converted from its digital value to an analog-equivalent current, which forces the actuator voice-coil motor to position the recording head precisely over the desired track.

Including the previous control state reduces the effect of the total system delay caused by microprocessor computation time, A/D and D/A conversion times, and actuator mechanical transportation lag. As a result, the actuator's ability to settle on a track following a track-seek operation is very fast, even though the total time delay for the file is roughly 60 microseconds, or about 25%, of the sampling period.

Track-Seek Mode

The track-seek control mode is used when moving the recording head from one track to another. The track-seek mode has two phases: the acceleration phase and the deceleration phase. The seek mode was quite complicated to design, as it includes factors such as non-linear actuator friction, voice-coil current rise time, mechanical resonances, and time required for the recording head to settle on a track. The acceleration phase provides an almost constant acceleration until a precalculated velocity has been reached. The precalculated velocity is generated from one of two carefully designed actuator deceleration-velocity profiles. A profile for short seeks and long seeks was developed to minimize seek time for each type of seek. The numbers representing the profiles are stored in the microprocessor and present the desired velocity during deceleration for any given distance from the desired track. In the deceleration phase, the servo control forces the actuator to follow the precalculated velocity profile (see Figure 2). The estimated velocity is subtracted from the desired profile velocity, and the resulting velocity difference (velocity error) is multiplied by a velocity-gain constant to yield the correct controller output to follow the velocity profile. (See Figure 4, equation 10, for the track-seek algorithm.)

The switch from track-seek mode to track-follow mode occurs when the recording head is 15% of one track-spacing away from the desired track.

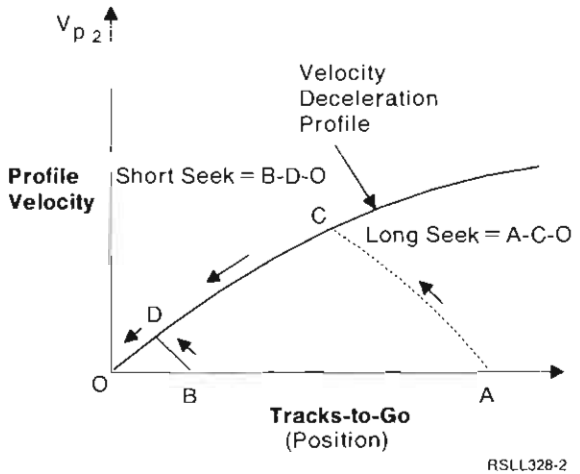


Figure 2 Velocity Deceleration Profile Versus Tracks-to-Go Used in the Track-Seek Mode

The deceleration-velocity profile is designed so that the head settles smoothly on the track in minimum time.

Self-Tuning of System Performance

Typically, a storage-device actuator's servo performance changes with time and environmental conditions. The disk units tuned to optimal performance during manufacture gradually become detuned with time and temperature. Some disk units use various compensation schemes to minimize this effect. Most of these schemes are typically designed for an average disk unit, in that all disk unit parameters are assumed to be at their average values. However, as hundreds of thousands of disk units are built, the average disk unit is just a statistical concept. It is therefore important to measure parameter variations for each individual disk unit while it is operating as part of a data processing system. Once parameter changes are identified, the gain coefficients can be automatically changed in the controller algorithm to again yield the optimal performance. Such a system is called an adaptive, or self-tuning, control system. Figure 3 shows a conceptual block diagram of an adaptive actuator servo-control system. The darkly shaded blocks are electromechanical components, while the lightly shaded blocks are microprocessor functions.

The 9332 has implemented an adaptive control system (patent-pending) to keep its controller tuned for peak performance over time and a range of temperatures. Two important values that can be measured or estimated online are: power-supply voltage, which is important during the track-seek mode, and a parameter called Γ_1, which is proportional to the loop gain of the closed-loop servo. Γ_1 is used to update the gain coefficients in both the track-seek and track-follow algorithms.

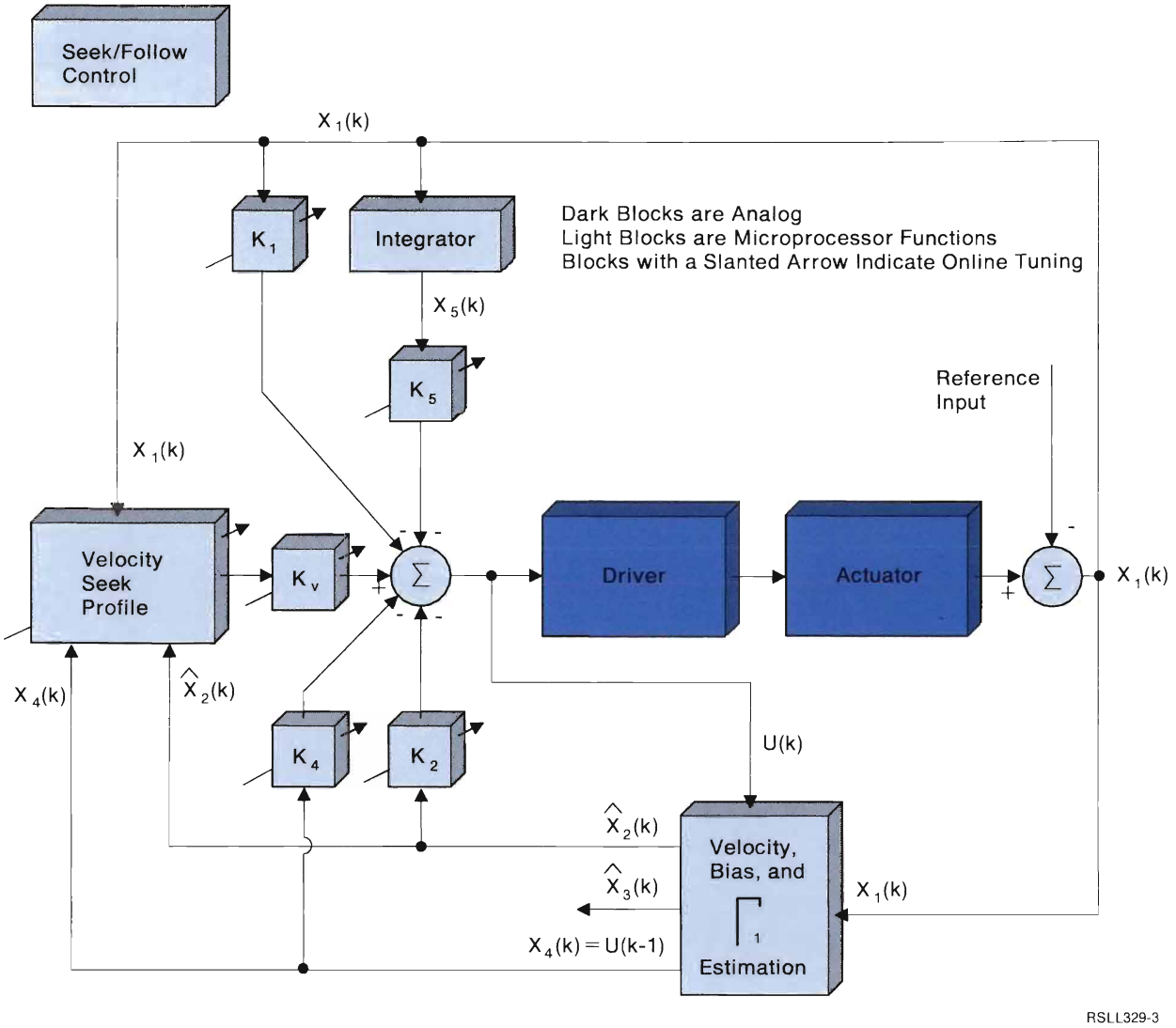


Figure 3 Conceptual Block Diagram of IBM 9332 Adaptive Actuator Servo Control

The power-supply voltage is measured by an A/D converter built into the microprocessor chip. The voltage, measured during initial and later online calibrations, is used to recompute the velocity profile table. Because a 1% change in power-supply voltage can introduce a 0.4% error in a generated profile velocity, the adaptive nature of the system is quite important by making the deceleration-velocity profiles independent of power-supply variations.

The Γ_1 parameter is slightly more difficult to estimate. The estimate can be obtained during acceleration when the actuator voice-coil driver is

supplying a known current and is not saturating. Position measurements and corresponding controls are obtained for several sectors, and estimates of Γ_1 are computed using a very simple algorithm (see Figure 4, equation 12). The estimates are then averaged to minimize the effects of noise and to yield a good overall estimate of Γ_1.

When Γ_1 has been estimated, the controller gains can be computed again. Then, with the updated controller gains, the track-seek and track-follow algorithms are modified for more optimal performance. A 1% change in Γ_1 causes a 0.5%

Actuator Model

The actuator model has five distinct variables called state variable X_1, X_2, X_3, X_4, and X_5. The five state variables have been normalized and are defined (with units) at k-th sampling instant as follows:

$X_1(k)$ - position (tracks)

$X_2(k)$ - velocity (tracks/sector period)

$X_3(k)$ - bias forces (equivalent tracks)

$X_4(k)$ - previous control (equivalent tracks)
$X_4(k) = U(k-1)$

$X_5(k)$ - integrated position (tracks)

$U(k)$ - control input (equivalent tracks)

The normalized digital state-space equivalent model of the sampled actuator with delay is:

$$X_1(k+1) = X_1(k) + X_2(k) + \Gamma_1 \cdot X_3(k) + \Gamma_{11} \cdot X_4(k) + \Gamma_{12} \cdot U(k) \quad (1)$$

$$X_2(k+1) = X_2(k) + \Gamma_2 \cdot X_3(k) + \Gamma_{21} \cdot X(k) + \Gamma_{22} \cdot U(k) \quad (2)$$

$$X_3(k+1) = X_3(k) \quad (3)$$

$$X_4(k+1) = U(k) \quad (4)$$

$$X_5(k+1) = X_1(k) + X_5(k) \quad (5)$$

The Γ_1's in equations (1) and (2) are functions of the actuator parameters, the sampling period T, and the delay time h.

Estimator Algorithm

The estimator state equations are:

$$\bar{X}_1(k) = X_1(k-1) + \hat{X}_2(k-1) + \Gamma_1 \cdot \hat{X}_3(k-1) + \Gamma_{11} \cdot X_4(k-1) + \Gamma_{12} \cdot U(k-1) \quad (6)$$

$$\hat{X}_2(k) = \hat{X}_2(k-1) + \Gamma_2 \cdot \hat{X}_3(k-1) + \Gamma_{21} \cdot X_4(k-1) + \Gamma_{22} \cdot U(k-1) + L_2 \cdot (X_1(k) - \bar{X}_1(k)) \quad (7)$$

$$\hat{X}_3(k) = \hat{X}_3(k-1) + L_3 \cdot (X_1(k) - \bar{X}_1(k)) \quad (8)$$

The definition for these new variables is given below; others have been given earlier.

$\bar{X}_1(k)$ - the predicted position at the k-th sector based on the information obtained from the (k-1)-th sector

$\hat{X}_2(k)$ - the estimated velocity

$\hat{X}_3(k)$ - the estimated bias force acting on the actuator

L_2, L_3 - estimator coefficients

Track-Follow Mode Algorithm

$$U(k) = -\hat{K}_1 \cdot X_1(k) - \hat{K}_2 \cdot \hat{X}_2(k) - \hat{K}_4 \cdot X_4(k) - \hat{K}_5 \cdot X_5(k) \quad (9)$$

where the states have been defined before. The controller gain constants $\hat{K}_1$, $\hat{K}_2$, $\hat{K}_4$, and $\hat{K}_5$ are calculated based on optimal track-follow performance and updated online.

Track-Seek Mode Algorithm

$$U(k) = -\hat{K}_v \cdot \left[Xp_2(k) - (\hat{X}_2(k) + \Gamma_{21} \cdot X_4(k))\right] - \hat{K}_5 \cdot X_5(k) \quad (10)$$

where some variables have been defined earlier. The profile velocity $Xp_2(k)$ corrected for delay is given by

$$Xp_2(k) = \text{PROFILE}\left[X_1(k) + \Gamma_{11} \cdot X_4(k)\right] \quad (11)$$

PROFILE = velocity deceleration profile function stored in the microprocessor table (Figure 2)

$\hat{K}_v$ = velocity profile control gain

Γ_1 - Estimator Algorithm

The estimated Γ_1 at the (k+1)-th period is a function of the second difference of the measured position divided by the sum of the two previous controls. Its discrete-time mathematical expression is given by

$$\hat{\Gamma}_1(k+1) = \frac{X_1(k+1) - 2 \cdot X_1(k) + X_1(k-1)}{U(k) + U(k-1)} \quad (12)$$

RSLL414-2

Figure 4 Digital Actuator Model and Algorithms for the Estimator, Controller, and Γ_1 - Estimator

error in the profile velocity at a given distance to the desired track. Such a velocity error increases the time it takes the head to settle on the desired track and, therefore, increases the average access time. For the track-follow mode, a 1% change in Γ_1 results in a 1% error for all of the controller-gain values used in the control algorithm, causing less than optimal tracking performance. Note that Γ_1 will change when changes occur in the actuator force-constant gain, current driver gain, position-error sensor gain, A/D and D/A gains, and so forth. Γ_1 is proportional to the product of all the gains above. Because Γ_1 is also an estimate of the low-frequency loop-gain, the controller algorithm is independent of dynamic changes in the low-frequency loop-gain.

Conclusions

The actuator servo control on the IBM 9332 Disk Unit opens a new dimension in storage device design. The control algorithms reside entirely in the microprocessor, making the servo digital in nature. Modern digital control theory was used for the design of this adaptive actuator servo-control system. The adaptive controller maintains uniform and predictable peak track-seek and track-follow performance over time and varying temperature. The results from servo performance measurement tests have been very promising. In the future, as very high-speed, low-cost microprocessors become available, more and more storage device functions will become self-tuning and adaptive to changes in disk unit components. The result will be disk units that are almost maintenance-free and always tuned for peak performance and reliability.

Acknowledgements

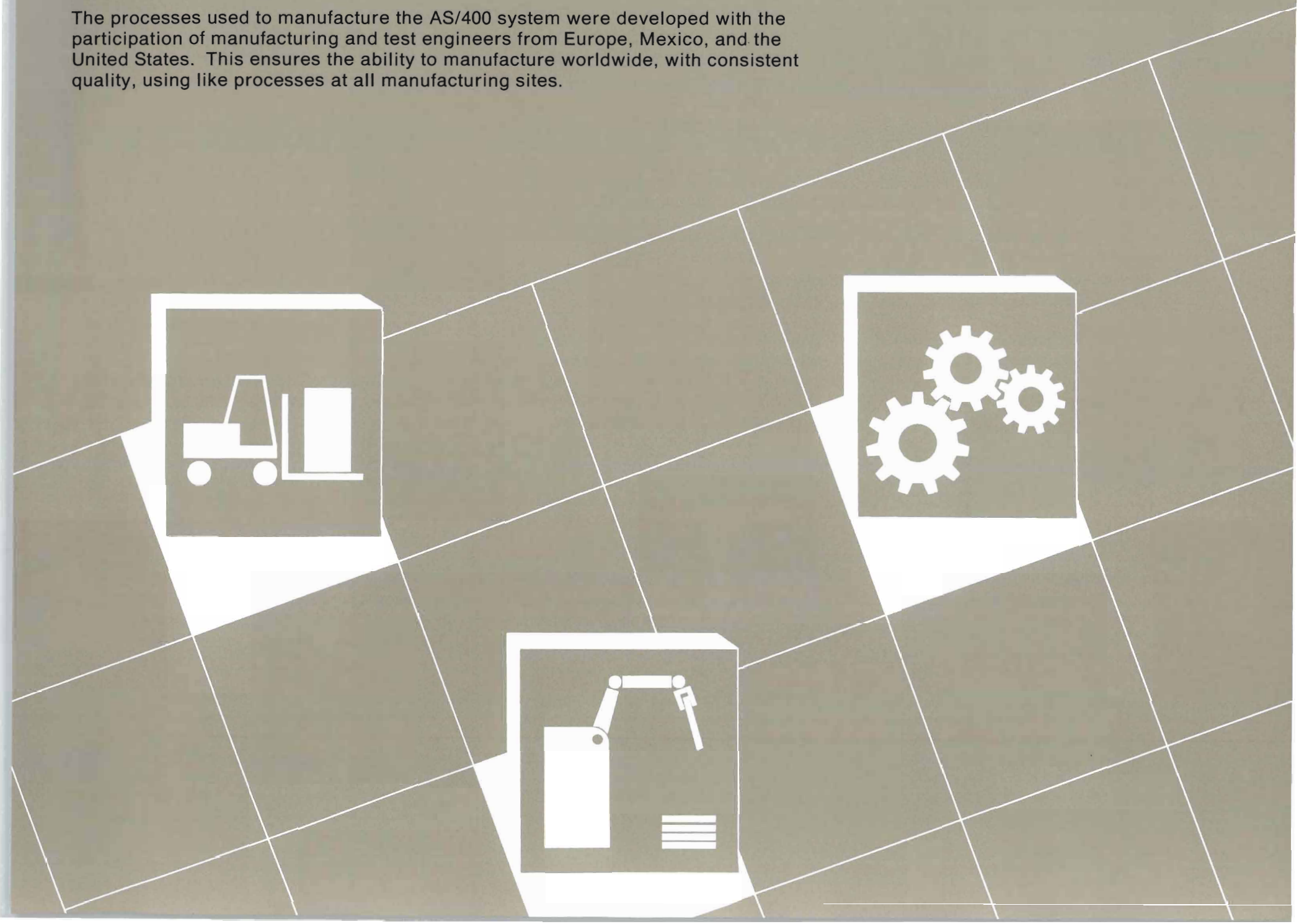
The adaptive digital servo control system for the IBM 9332 Disk Unit was developed in a team effort. The implementation of such a system requires dedicated microcoding and mechanical and electrical design efforts with uncountable hours of testing. Credit and recognition for the servo control should go to Michael C. Stich, Todd B. Anderson, John B. Resman, and many more. Special gratitude goes to our consultant at IBM Rochester, Dr. G.F. Franklin, Stanford University, Stanford, CA.

References

1. Oswald, R.K., *Design of a Disk File Head-Positioning Servo*, **IBM Journal of Research and Development**, Volume 18, 1974. 506.
2. Franklin, G.F. and J.D. Powell, **Digital Control of Dynamic Systems**, Reading MA: Addison-Wesley Publishing Company, 1980.
3. Stich, M.C., *Digital Servo Algorithm for Disk Actuator Control*, **Proceedings of Conference on Applied Motion Control '87**, June 16 - 18, 1987, Minneapolis, MN. 35 - 41.

Manufacturing

The processes used to manufacture the AS/400 system were developed with the participation of manufacturing and test engineers from Europe, Mexico, and the United States. This ensures the ability to manufacture worldwide, with consistent quality, using like processes at all manufacturing sites.



The Flexible Manufacturing System

Describes the flexible manufacturing system designed to efficiently produce all models of the AS/400 system.

Donald L. Conroy

Introduction

The flexible manufacturing system is a low-cost production facility capable of producing any configuration of any model AS/400™ system, to a customized order, with no set-up time required between models.

With an emphasis on simplicity, this production facility is manually oriented, provides expansion capability, and uses a floor-control system driven by IBM PERSONAL COMPUTER AT's®. This low-cost combination provides maximum flexibility for AS/400 manufacturing. To simplify the process, a concentrated effort was placed on strategic design, early manufacturing involvement, continuous flow manufacturing, and computer-integrated control. Other factors critical to reducing complexity were modular product design, minimum part content, and reduced line storage.

Strategic Design and Early Manufacturing Involvement

The manufacturing team for the AS/400 system was organized while the product's design was still being formulated. The team's mission was to assist in developing a product from the perspective of ease of manufacturing and reduced product cost.

The modular design of the system was a direct result of this relationship. By designing pluggable subassemblies, the capability to customize the product was increased, with little or no increase to assembly complexity. The uniform design of the subassemblies allowed the use of standardized parts' storage areas so if customer model demands shift over time, the floor layout can remain relatively unchanged. Set-up time between models is eliminated, because different models require the same basic assembly operations. Minimizing the number of parts simplified both assembly and parts storage. Snap covers and thumb screws reduced assembly time and tooling requirements. Planning such as a standardized cable design reduced storage-space requirements on the manufacturing floor. Manufacturing emphasis during the design stage clearly increased assembly flexibility and reduced capital investment in the manufacturing process.

This was only one piece of the early manufacturing involvement process, however. While recommending design changes to the product development group, the early manufacturing involvement design team was also feeding information back to an early manufacturing involvement process team. This team consisted of lead engineers representing process, systems, procurement, distribution, and production control. Emphasis was placed on an in-the-door, out-the-door philosophy from supplier lines to shipping and installation. When product design information became available, each of the engineers would examine the parts. Suppliers were given early views of the part designs and recommendations were fed back to the development laboratory. Every part was examined for manufacturability, delivery lead time, tooling requirements, packaging/shipping expense, and commonality. The results and recommendations were continually rolled up and fed back to development for final design consideration. Manufacturing process evolution coincided with product evolution.

Continuous Flow Manufacturing

The modular AS/400 design made it possible to manufacture highly customized orders in a production environment. The number of assembly steps required was significantly reduced, which enhanced process flow. Still, the number of parts required at the assembly stations was very large because of the variety of system offerings. Large automated storage and retrieval systems were considered a parts-containment requirement. Logistics control of the high-volume, customized process appeared to require significant programming effort and hardware cost. As an alternative approach, the manufacturing team began to look at continuous flow manufacturing (CFM) as a solution to the parts-control problems. It was examined from two directions: the flow within the manufacturing process from work station to work station (MICRO-CFM), and the flow external to the manufacturing process of parts from suppliers and systems to customers (MACRO-CFM). Implementing CFM reduced parts inventory and eliminated the use of complex logistics systems to maintain station-to-station parts control.

Computer-Integrated Manufacturing

The computer-integrated manufacturing team, consisting of manufacturing engineers, distribution engineers, and systems analysts, was formed along with the design and process teams during the early manufacturing-involvement cycle.

Their mission was to channel worldwide order inputs into a data base of system parts, select the parts based on the customer order, and provide customized assembly instructions for the technicians to assemble the order. Interaction was the uppermost computer-integrated manufacturing activity. Adjustments to the process were identified to potentially reduce floor-control systems architecture requirements. The system design was looked at and adjusted to increase the flexibility of the physical assembly process. The team adapted the logistics systems during the product and process development cycle to ensure a solid transition from design to production.

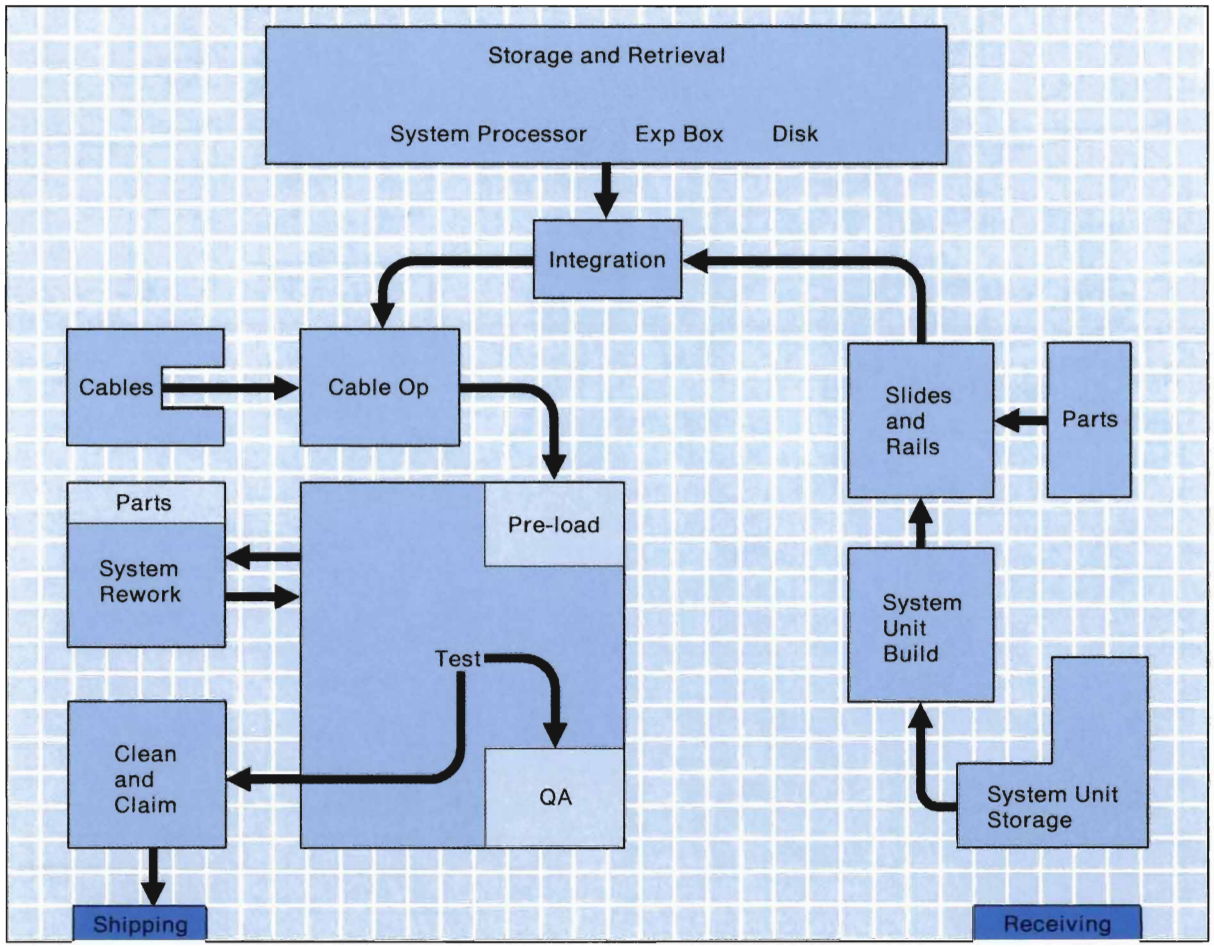
The Flexible Manufacturing System

A traditional manufacturing layout is designed around the product and concentrates mainly on product shipment. The flexible manufacturing system used the process flow, not the product, as the key design point. The layout was conceptualized before the product designs were received in manufacturing.

Two major elements make up a process flow design: parts coming in and product going out. When continuous flow manufacturing is not used, parts flow can be considered a minor design point. Problem part locations are handled by adding more storage space to the layout and refilling stock less frequently. The cost is increased space and inventory expense. With CFM, parts storage is minimized. This means less dollars invested in plant and equipment, but parts must be replenished at more frequent intervals. Because of this frequency, a good parts flow design produces significant savings in time and labor.

Frequent restocking, unless properly planned for, can severely hinder product flow, which in turn reduces output. The flexible manufacturing system uses a **U**-shaped design to facilitate flow (see Figure 1). Stations are set in place along the U, with parts stored on the outside and product flow set on the inside. The outside parts storage allows easy access to the storage areas without interrupting the work in process. The inside product flow keeps work stations close, allowing visible management and minimum product movement.

The U shape focuses the flow at the shipping and receiving dock. Parts are brought into the receiving area and unpacked before being taken inside the U. This eliminates congestion on the manufacturing line caused by used packing material kept on the line. The parts are moved to the manufacturing line and placed in highly visible storage areas. CFM technicians visibly monitor these storage areas and replenish them when quantities reach predetermined levels. Small parts are manually transported to the line. Larger parts, plus completed systems, are delivered by an Automatic Guided Vehicle System to the front of the U and the CFM technicians stock them



RSLL407-2

Figure 1 AS/400 Manufacturing Process Flow

Figure 2 Automatic Guided Vehicle Delivers Completed System to Distribution Center

manually from there (see Figure 2). Manual parts delivery provides maximum flexibility to the process design. Work station locations are not governed by conveyor-spur locations and fork-lift aisles are not needed. Shipping damage and scheduling problems frequently encountered with fork-truck deliveries are eliminated. The advantage of removing fixed restrictions such as these will be more apparent in the future when new models are added to the existing system configurations. The process can be adjusted based solely on product and parts flow without having to work them into existing, inflexible process hardware.

The assembly work stations themselves consist of modular work surfaces strategically placed on the inside of the U. Modular stations allow the work area to be customized not only for the product, but for the individual assembly technician as well (see Figure 3). This is important in a CFM environment where the workers share tasks. Each worker can adjust any station for size, height, and preference. The U-shaped flow allows complete visibility of the entire process. Bottlenecks can be visibly identified and workers can leave their stations immediately to assist the backed-up area until it is on schedule with the rest of manufacturing line.

Heavy subassemblies that could not be handled manually were grouped into one assembly operation, and a mechanized transfer system was designed and installed. This transfer system is a fixed work station, but by grouping the heavy tasks into one operation, the requirement for fixed work stations was restricted to this one. This anchored station was made as flexible as possible to be capable of handling any model in any configuration. This required the transfer system to operate on demand for an individual order. The transfer car was designed to pick up specified subassemblies and deliver them to a central lift device where the technician could install them into the product. The transfer car is fed by gravity-feed conveyor spurs, limited to a maximum of four for each subassembly. The size of four was selected for easily visible parts management. CFM technicians monitor the spurs to ensure adequate supply; a flashing light indicates an empty or malfunctioning spur. The base transfer car system size was determined by the number of subassemblies in the current AS/400 offering plus six extra spurs to accommodate fluctuating demand and model mix. The car's drive cable is longer than the existing track so that future requirements for additional spurs could be served by simply lengthening the track. The U-shaped process was placed at one end of the building. If demand exceeded the expansion capability of the existing process, the modular work stations could be duplicated (in a mirror image) at the other end of the transfer system. This would essentially double the maximum capacity of the line without adding any new fixed equipment. The expense of doubling the capacity would be limited to the low-cost, modular work stations and some minor modifications to the transfer system.

The assembly process flow is controlled by sets of **kanban** squares. (Kanban is a term for a marked area before and after each work station.) A kanban is defined for each part that enters a station and each part that leaves it. The input

Figure 3 A Modular Assembly Work Station

kanban for one station is the output kanban from the previous station. Every square has a maximum limit; if a square contains the maximum number, no more parts are allowed to go into that area until some are used up. If the squares in front of the station are empty, a bottleneck must exist in some operation prior to that station. The operator would leave the station and lend assistance to the backed-up area. If the squares behind the station are full, a bottleneck exists after that operation. The operator stops work at the station and again helps out the problem area. When a square is empty after the work station and full before the station, the operator continues to work. This simple concept controls the entire flow of products through the manufacturing line. Our flexible manufacturing system uses a kanban size of two. The small size allows problems to surface immediately and be resolved. If a batch of defective parts enters the process, the maximum number of units that can contain these parts when the problem is discovered is two per station. The problem is contained within the process. Work-in-process rework costs are negligible. The defective parts are replaced and the process flow continues on. CFM process control is as effective as any of the complex systems-architecture designs evident in either process floor-control systems. Cost of the kanban process is essentially limited to a few rolls of colored tape used to mark the kanban squares on the floor.

Personal computers direct the assembly at each work station and control the transfer car, allowing it to deliver the correct subassemblies for each order. Personal computers also provide integrity to the CFM concept by ensuring that work does not begin on an order until all previous steps have been completed. At each station, the personal computer displays a list of parts and their subsequent locations within the unique order that is about to be processed. This display of parts and locations is limited to the tasks required at that particular work station. Thus, the operator is provided with instructions that allow complete customization of the product on an order-by-order basis. At the transfer car station, the personal computer displays the information to the operator, and, at the same time, directs the transfer car to deliver the subassemblies to the operator. The

assemblies are delivered to the operator in the same sequence they are to be installed on the order. The segmented order data is passed from work station to work station until the completed product is finished for delivery at the work station located closest to shipping dock.

This process provides a continuous flow of customized systems with maximum output, minimum expense, and no set-up time between models.

Conclusions

Simplicity is a complex engineering challenge. Minimizing parts storage encompasses certain risks when parts are ordered from a single supplier. Monitoring and managing a simplified system is more demanding than running a system loaded with parts and capacity. Still, the efficiencies generated in product flow and inventory savings outweigh the risks. By addressing these concerns early and concentrating on simplicity and flexibility, the flexible manufacturing system resulted in a highly efficient process for customizing AS/400 products.

Manufacturing Card and System Tests

Describes how early involvement and enhanced testing enabled the manufacturing group to deliver high-quality products at a lower cost.

Robert W. Lytle, Donald L. Beck, Mark W. Hansen, and Gary L. Kearns

Introduction

The manufacturing test objectives on the AS/400™ system were very simple: reduce the time and cost of testing yet deliver a system that meets the most stringent quality criteria of any system ever shipped from IBM, Rochester, MN. Because our traditional test philosophy would not meet the requirements for a shorter product cycle, a higher-quality product, and lower manufacturing costs, new methods were introduced to the development and manufacturing processes.

First, manufacturing engineering became involved in the development stage of the product to help ensure that a stable design was delivered to manufacturing. Manufacturing engineers stress-tested logic cards during early engineering tests. Second, a functional card test was used to reduce the number of test steps and enhance the effectiveness of the test; in a functional test, the cards are tested in a simulated systems environment. With a stable design and highly efficient card testing, the final system test for the product became a verification of the final assembly process, rather than just a screen for defects.

Early Manufacturing Involvement with Stress Testing

Prior systems were released to manufacturing when functional specifications were met. Later, in production, stress screening indicated that too many parts would not function within the system specifications. This resulted in excessive scrap, rework, and retesting.

To prevent this from happening on the AS/400 system, early stress tests were performed on logic-card assemblies from early engineering prototypes through the final design. An extra margin of safety, or guardband, was verified in this testing, ensuring that later manufactured parts, obtained from multiple worldwide sources, would operate properly through the full system-specification range.

Guardband testing demonstrates performance capability beyond normal system specifications. It involves subjecting early design hardware to extreme operating voltages, temperatures, and oscillator frequencies to determine the actual functional limits of a given design. By doing early testing while development engineers were still heavily involved, key technical people were available to diagnose and repair potential problems found during the tests. The early detection of problems allowed time to modify designs and improve manufacturing quality before volume manufacturing began. This approach eliminated the need to depend on a production stress test for the life of the product.

Each logic card was stressed to at least 5°C beyond the upper and lower temperature limits specified for the component technologies used on each card (see Figure 1). Some cards were tested to as high as 90°C. The voltage stress limits were a minimum of ± 10% beyond the component nominal specification limits. Where feasible, oscillator frequencies were also varied. To ensure thoroughness, each card was tested using a four-

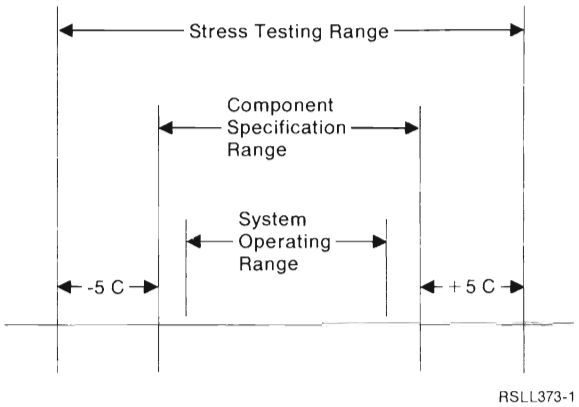


Figure 1 Test Limits

or eight-corner test matrix. For example, one combination of an eight-corner matrix might be high temperature, low voltage, and a fast oscillator.

Software test tools, such as system exercisers and simulators, were used to run the cards during the testing. A temperature stress chamber environment was used to stress test each individual logic card before the cards were integrated into a system. Variable power supplies were used to apply voltage stress to the cards. Test data results, including timing measurements, were taken as each card was stressed. This data was tracked closely to ensure problem resolution.

The results of early stress testing were significant. Of eight different card types tested, three failed under some combination of stress conditions. These problems were fixed through design

improvements or module changes well ahead of high-volume manufacturing. When more logic cards became available, additional cards of each type were tested over a period of several months to see if any module variations, due to different manufacturing batches of cards or components, were found.

In addition to testing individual cards, a system-integration stress test was performed on the first working systems. The integration stress test was performed with temperature and voltage variations similar to the individual card tests. This again led to early problem detection and resolution.

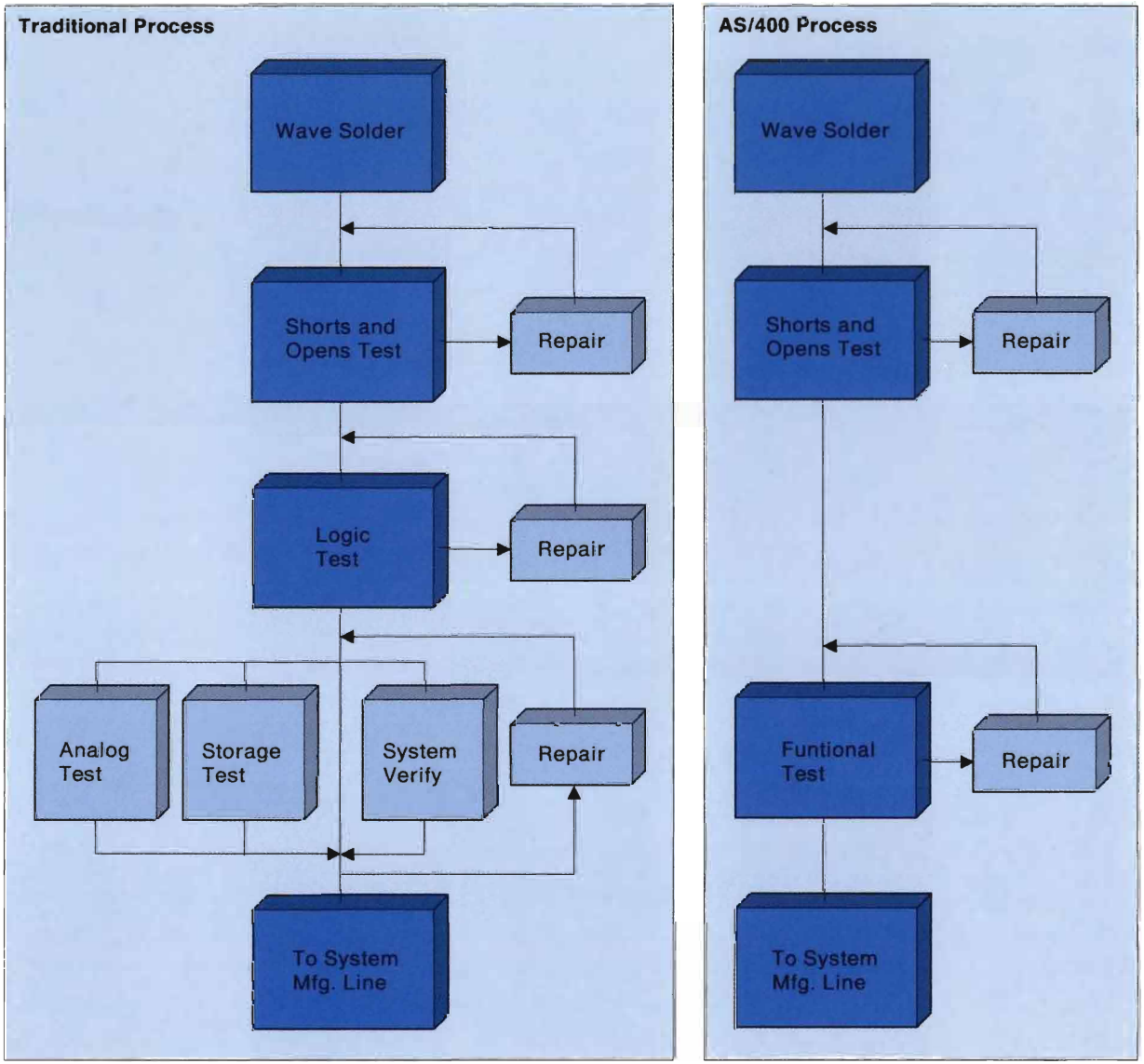
Production Card Test

In addition to the advances in early stress testing, significant improvements were made in the production testing of logic cards.

Single-Step Functional Test

Figure 2 demonstrates the simplified test steps with functional testing. Previously, specialized instruments tested different card types. The AS/400 card functional test consists of one step using one standard test instrument.

The functional card test is more effective than the traditional stuck-fault test (see Figure 3). During a stuck-fault test, the card sees patterns of 1's and 0's, with no functional meaning, applied at speeds much slower than in the actual system. During a functional test, the card receives the same signals and instructions it would see in an actual system running in a customer environment. The functional test takes advantage of the microprocessors on the various AS/400 logic cards. The micro-processor tests itself, all the logic contained on the card, and then all external interfaces to the card. In addition to the quality improvements, savings were realized in engineering, maintenance, manufacturing resources, and inventory.



RSLL374-3

Figure 2 Logic Card Test Process

Selective Stress Testing

On previous products, stress tests were performed on the complete system, stressing all logic cards to the same limits. Although this testing is beneficial, the AS/400 functional test subjects individual cards to stress parameters optimized for each card type.

Early in the production phase of AS/400 logic cards, test results were used to produce stress

Figure 3 Logic Card Being Functionally Tested

profiles tailored to the particular failure modes of each type of card. Using the profiles, the automated tester subjects the logic cards to more stringent tests, resulting in a higher-quality product at a significant cost savings.

Real-Time Data Collection

Once the optimal tests were established, real-time data collection was used to monitor the test process. Components not meeting their committed quality levels were found immediately, eliminating unnecessary testing. The end result is that data collection provides the information necessary to continually improve the efficiency of the process and the quality of the AS/400 system.

Final System Test

The final system test of the assembled AS/400 system is one additional safeguard to prevent shipping any defective parts to a customer (see Figure 4).

Test results on previous products showed that only a few components had a high failure rate after the first few hours of final test. Quality engineering established a requirement on the AS/400 system that all parts arriving on the final manufacturing line must already be fully tested and of shippable quality. To ensure this requirement was satisfied, extensive testing was done during the early manufacturing build cycle. A large sample of systems was subjected to a very long final test to ensure that systems that pass the standard final test will continue to function correctly.

With this requirement in place, the final system test was done primarily to ensure the system was assembled correctly. No extended run-in or burn-in was needed to improve the reliability of the system. This final test process resulted in a higher-quality product, along with significant savings in manufacturing costs.

Figure 4 Final System Test

To verify the effectiveness of early stress testing, an audit stress test on a small sample of systems was implemented to ensure once again that quality levels were met.

Conclusions

The methods described represent significant changes in the IBM Rochester manufacturing test strategy, compared to the methods used on previous products. The changes were driven by requirements for a shorter development cycle, higher quality, and lower costs. Early manufacturing involvement was a key element in shortening the product cycle, because design problems were uncovered and repaired before manufacturing began. The simplified testing in card and system manufacturing improved the quality of the assembled product at a significant cost savings. The AS/400 product represents a new milestone in manufacturing technology for IBM Rochester.

Disk Unit Manufacturing Process

Describes advances in manufacturing processes and technology used to assemble disk units for the AS/400 system.

John T. Costello, Gary L. Landon, and Thomas J. Warne

Introduction

The assembly and test of rigid-disk storage units is a marriage of high technology and precision components in the manufacturing process (see Figure 1). The assembly consists of magnetic heads, magnetic media, a data channel, and an enclosure. The disk units are used for information storage and retrieval for computer processing.

Unique techniques are used to merge heads and disks on the Disk Unit (Feature #6100) used in the 9404 System Unit. And, on the 9332 Disk Unit, the disk unit's electronics and microcode perform the surface analysis tests on itself. New supply logistics (materials support and flow), assembly process control, and disk unit testing techniques provide efficient, high-quality, and low-cost manufacturing and subsequent delivery of extremely reliable disk units for the AS/400™ system and other computer systems.

In addition to the design, the manufacturing process is a key ingredient for producing a reliable disk unit. IBM, Rochester, MN, uses continuous flow manufacturing (CFM) to optimize production and statistical process control techniques to ensure high quality in shipments.

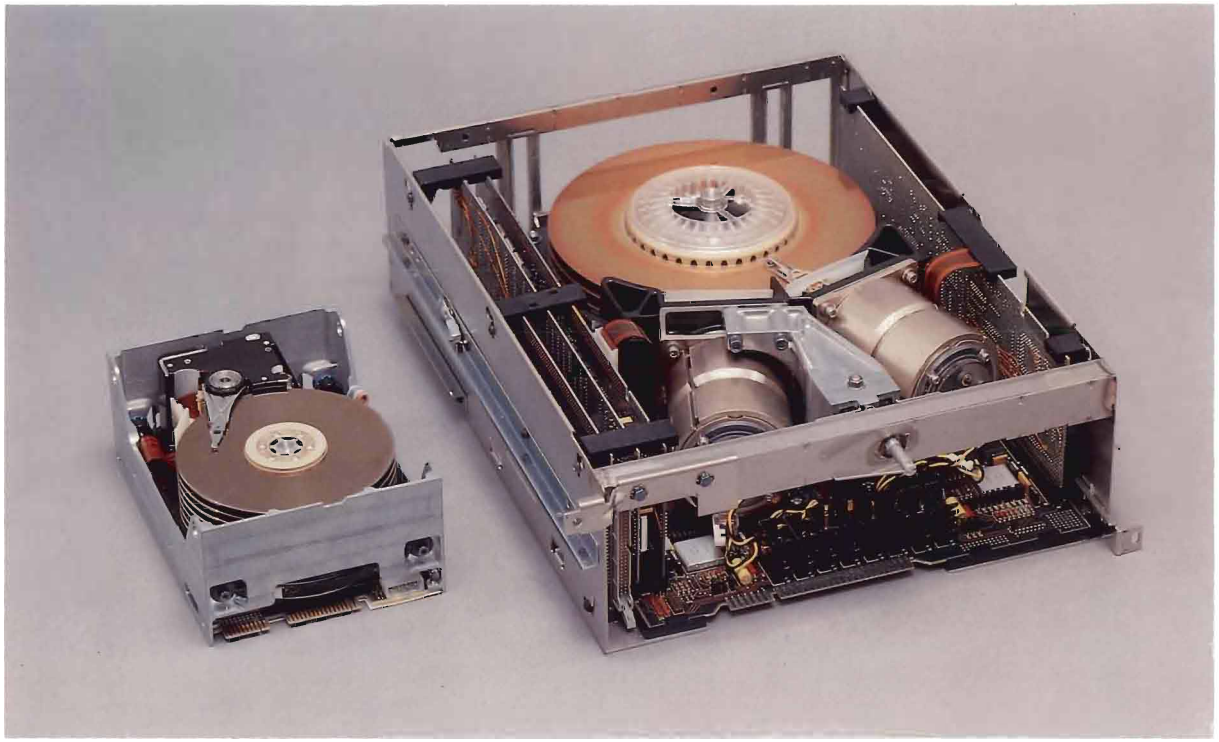


Figure 1 5¼″ and 8″ Rigid Disk Storage Files with Covers Removed

Logistics Support

CFM is our strategy now and in the future because of the significant advantages achieved using this manufacturing philosophy. Assembly and test processes follow CFM concepts that are centered around just-in-time manufacturing, more commonly known as JIT. These concepts focus on elimination of excesses, total people involvement, and total quality control. To support CFM processes in the manufacturing plant, a continuous flow of incoming, defect-free precision components, assemblies, and supplies is required.

At the product design stage, an early manufacturing involvement purchasing team was established to work with suppliers, development, and manufacturing. The team established supplier selection and qualification criteria that includes using CFM and statistical process control. This team then involved the suppliers at the design level, allowing for better manufacturability early in the program.

High-cost purchased parts are frequently delivered to the manufacturing line by way of the plant receiving dock, or pulled to the plant dock from suppliers through a signal (phone call, Electronic Data Interchange, or other method) as

required. Parts that are delivered directly, or pulled through receiving, bypass inspection and warehouse stock areas. The parts are taken to the manufacturing line for immediate use, or are placed in work-in-process storage areas next to the assembly line. (For more information, see the article *Electronic Data Interchange*.) The parts ready for use are placed on a carousel. As parts are requested, they are moved from the carousel through a cleaner and assembled, minimizing potential contamination. Work-in-process inventories are kept quite small by storing them in highly visible storage areas where they can be readily managed.

After the disk units have been assembled and tested, they are placed into a work-in-process transport cart to be moved to the systems manufacturing line or the shipping dock. The disk units are pulled, as needed, to the systems area from manufacturing on a daily basis, or as needed. The signal to replenish inventory at the system areas is empty carts. The number of carts is kept low to minimize work-in-process inventory between disk unit manufacturing and the using areas.

Assembly Process Control

In the disk unit assembly process, several major activities are used to control and monitor product flow, including CFM, a manufacturing control system, statistical process control, and automation.

Because the Disk Unit (#6100) used in the 9404 System Unit is small, operations are placed close together to allow manual transfers. Placing operations close together has reduced space requirements and the need for material handling systems. Pull logic is a CFM technique used to control assembly build operations and production line flow. Assemblies are pulled from upstream operations as they are used; inventory is not allowed to build up waiting for use by downstream operations. Disk unit assembly improvements resulting from CFM pull logic are management by sight, reduced work-in-process, and inventory replenishment based on consumption. CFM concepts were implemented during the design stage of this disk assembly program. (See the article *The Flexible Manufacturing System* for additional information.)

The manufacturing control system is a large central computer complex that is attached through the network control unit and coaxial cables to each test cell, display station, and process computer within the manufacturing area (see Figure 2). The manufacturing control system is used primarily for tester control, process sequencing, and data collection. This data provides a history of all major events in the

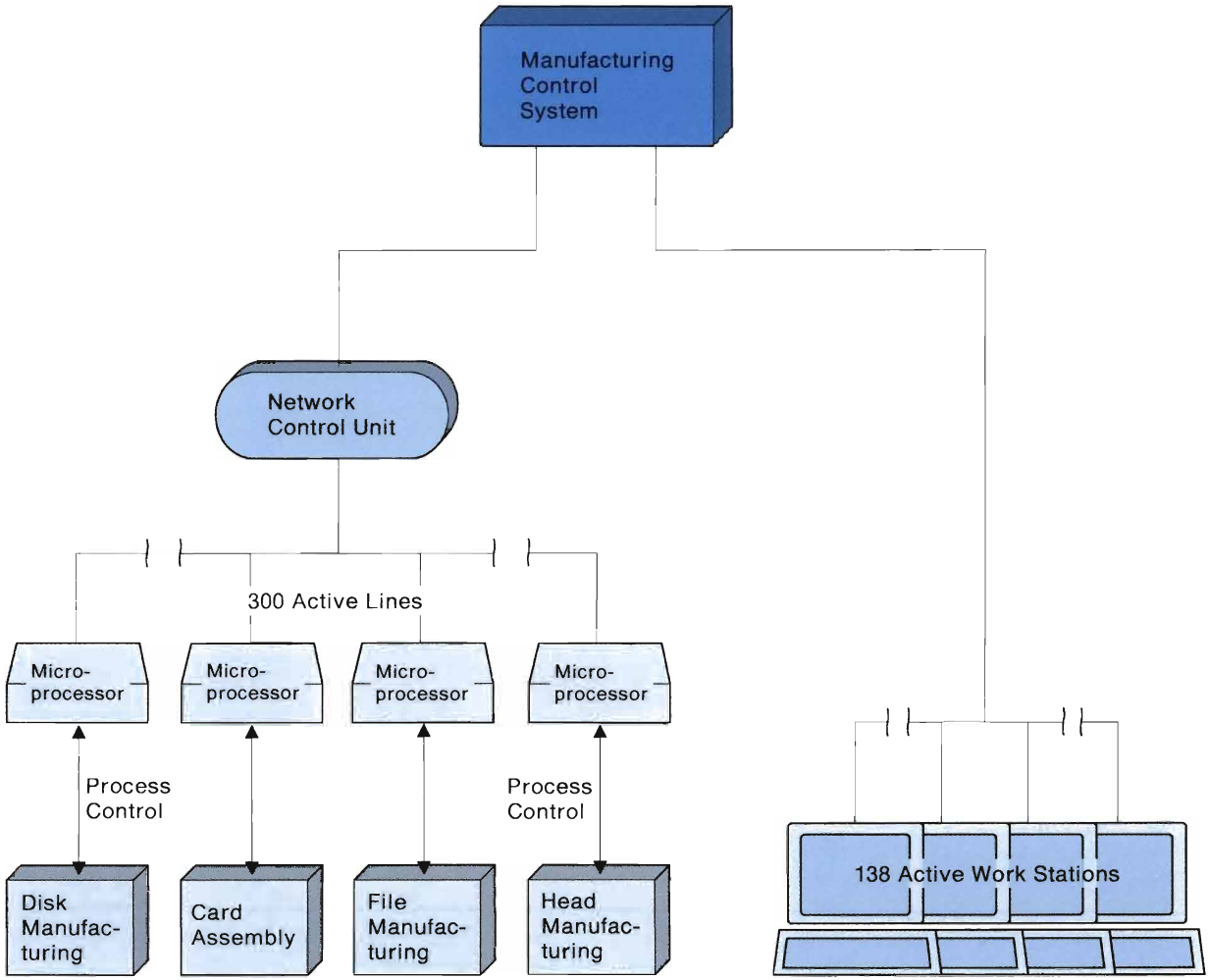


Figure 2 The Manufacturing Control System

RSLL420-0

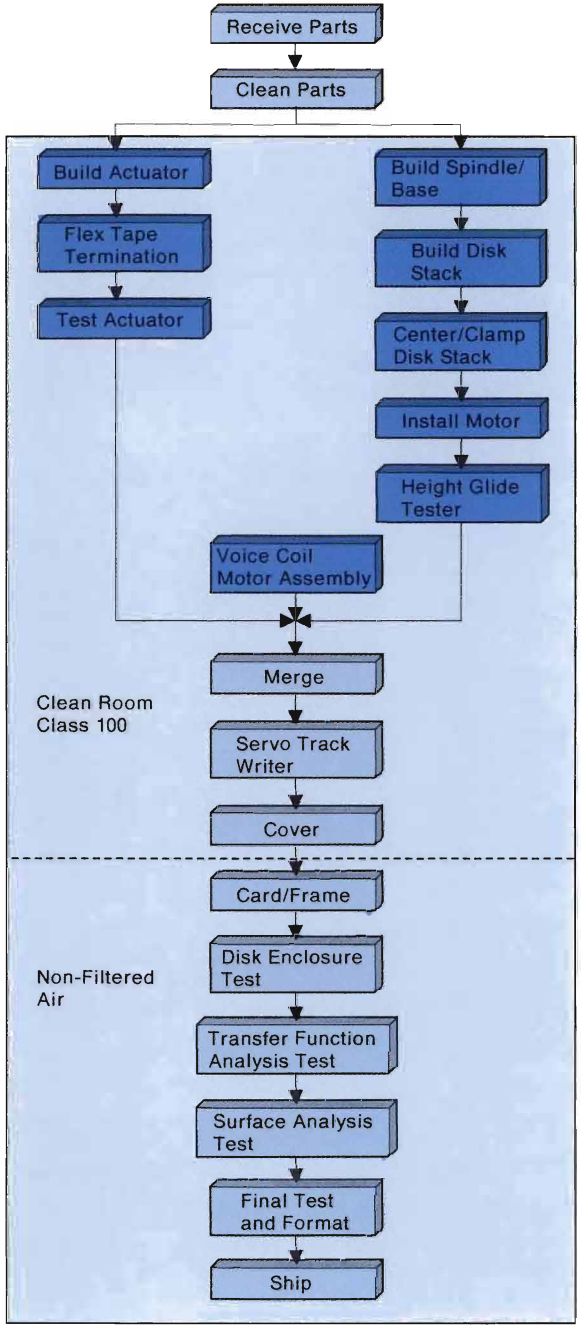


Figure 3 Process Flow

manufacturing process. The control system directs the flow of parts and ensures that the correct build and test sequence is followed. The test and process data retrieved by the system from each test and process station is used for engineering analysis of yields and process performance. Process data is also saved in permanent storage for later reference.

Another method used for process control is statistical process control. This is a statistical method used to evaluate objectively the performance and variability of manufacturing processes. The manufacturing control system collects this statistical data for analysis. Control charts are automatically generated to provide timely feedback to engineering and manufacturing on specific key parameters and processes. These charts identify trends so defects can be anticipated and prevented and process variables reduced over time.

Process Flow

The disk unit build processes begin on two distinct lines, the actuator build line and the spindle build line, which merge to form a device-enclosure line (see Figure 3). The device-enclosure assembly proceeds through this line into testing.

On the actuator line, arms are stacked, aligned, and clamped to form the actuator body. Next, electrical connections are made between the head coil wires and the arm electronic-terminating pads, using a solder reflow process. Solder deposited on the terminating pads is heated locally to a liquid state, which allows the wire connection. Then, the actuator assembly is tested for electrical continuity and sent to the merge operation.

On the spindle assembly line (see Figure 4), a bearing is placed into a sleeve that is inserted into the spindle bore. This subassembly is then bonded into place using ultraviolet (UV) light to cure the adhesive. This new UV process reduces adhesive cure time, thus lowering work-in-process inventory buildup. Disks and spacers are then stacked onto the spindle hub by a robot to form a base assembly. The disk stack is centered to ensure that all disks are concentric with the spindle, then clamped and passed to height-glide testing.

The primary purpose of height-glide testing is to ensure the disk spindle assembly can be merged with the actuator assembly without damage (see Figure 5). Disk surface asperities and

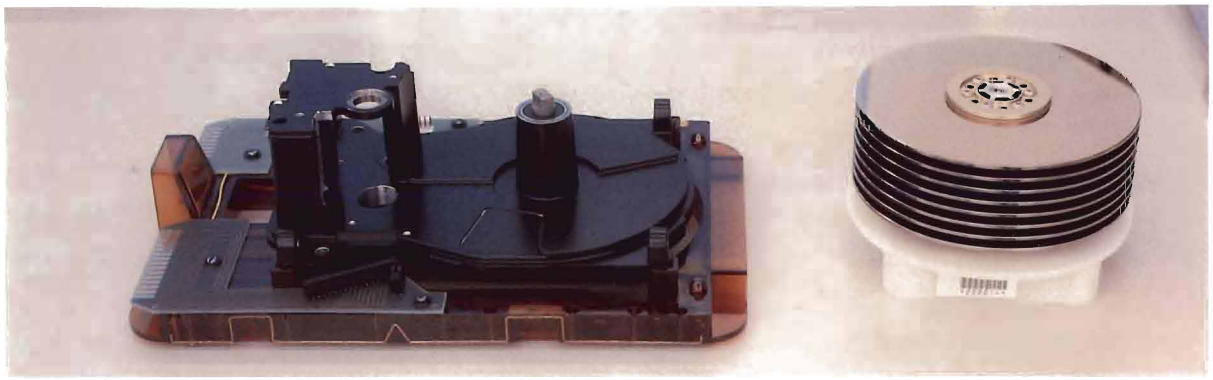


Figure 4 5¼″ Disk-Stack Assembly Tool

imperfections, static and dynamic disk height, and spindle bearings are checked to verify they meet specifications.

In the merge operation, the base assembly, with its disk stack, is brought together with an actuator assembly to form a disk enclosure. Care is taken not to damage the heads or disk surfaces, or to generate any contamination, as this could result in loss of data, errors in read or write, and even head crashes. Due to the closeness of disk spacing, special tools and techniques uniquely float the heads onto each disk during merge. This method minimizes head and disk damage due to contact and prevents generation of contamination.

After the merge operation, servo tracks are written on each disk unit (see Figure 6). On the Disk Unit (#6100), data heads are used to servo-write and read back track positions. On the 9332 Disk Unit, for manufacturing throughput, special heads write the servo tracks and the disk unit data heads provide read back of track position. The servo data is written on a dedicated surface or on each data surface prior to each data sector boundary, or on both, if necessary for file performance. A laser feedback mechanism is used to position the heads at the correct track spacing. These tracks must be precisely written, both radially and on the circumference, so that data can be written and retrieved without interference from information stored on adjacent tracks. (For more information, see the article *Digital Servo Control for Disk Units*.)

Before the disk enclosure leaves the clean room, it must be sealed to protect it from outside contamination. Class 100 conditions (meaning that less than 100 particles of 0.5 micrometer size are found per cubic foot of air) are maintained in the enclosure and a pressure test is done to ensure the cover is sealed properly.

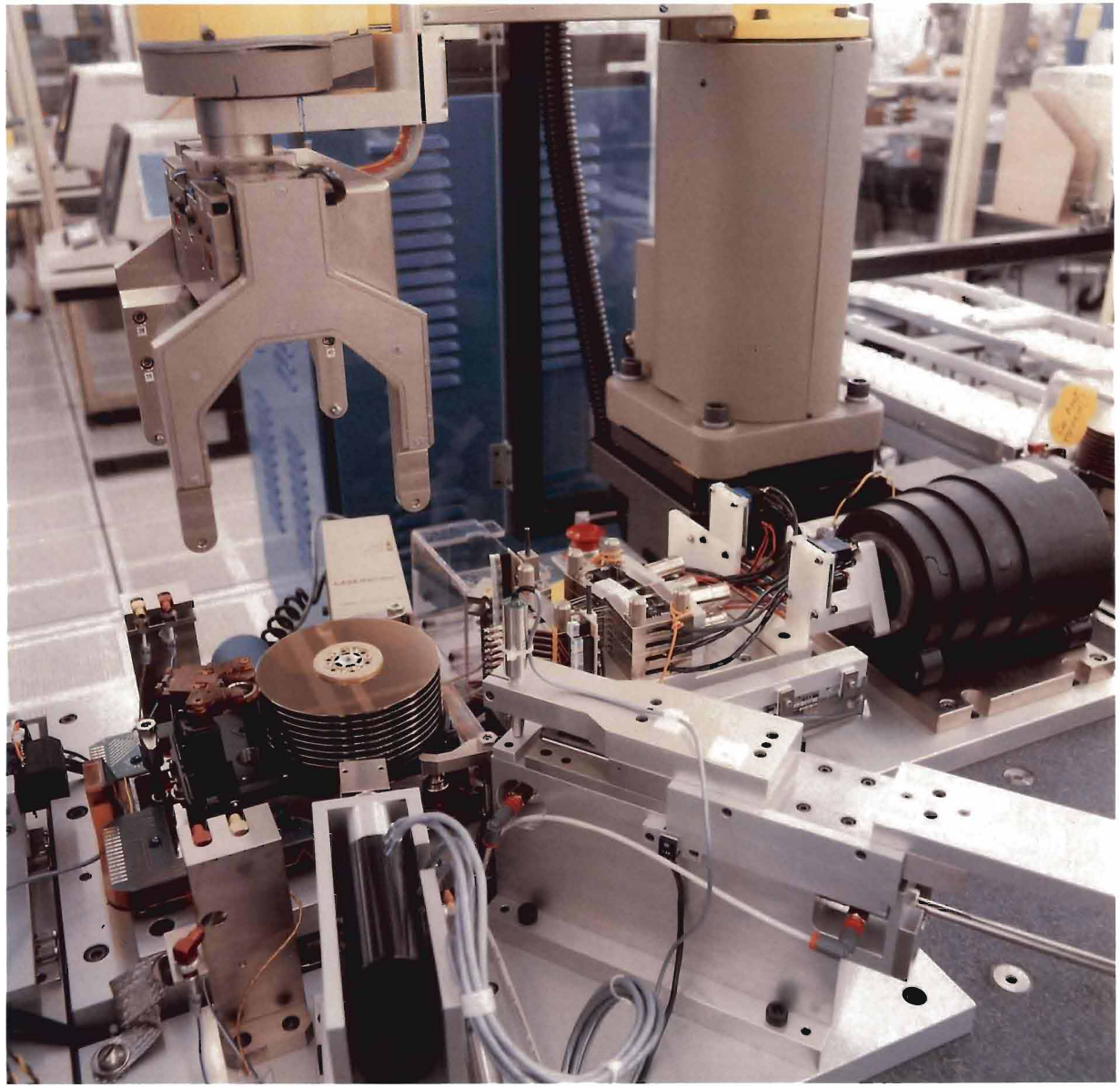


Figure 5 5¼″ Height-Glide Tester

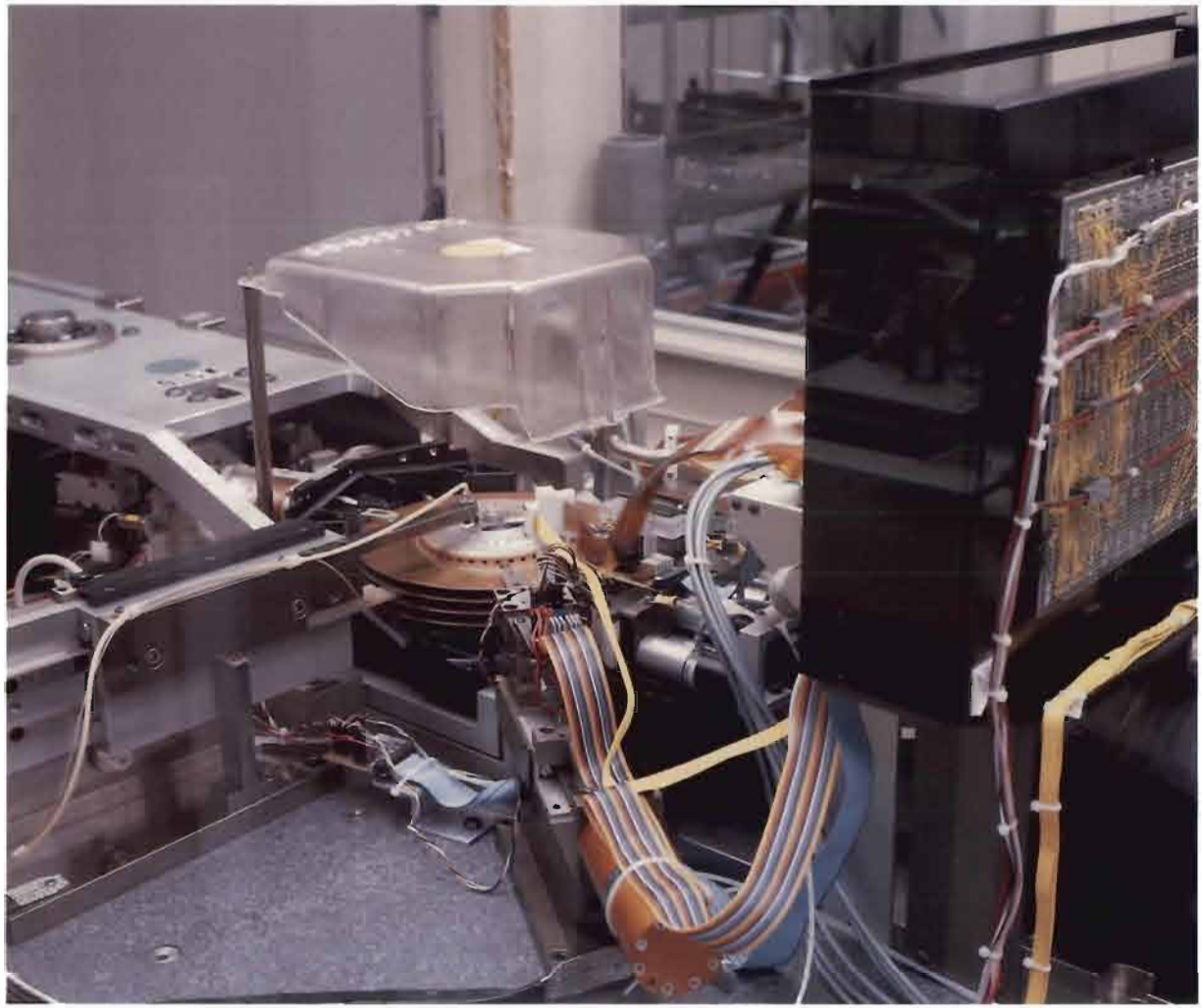


Figure 6 8″ Servo-Track Writer

After the enclosure is moved from the clean room, the frame and logic card are attached to complete the disk unit, and the assembly is tested to ensure proper function.

Precision Handling

Precision handling techniques are used throughout disk unit assembly, including disk handling, disk-stack assembly, head-to-disk assembly, and merge. A work-in-process carrier is used to transport the disk unit through the assembly process. The carrier protects it from damage, contamination, and electrostatic discharge (ESD). It also serves as an interface to the tools and testers. The disk (magnetic media) is susceptible to damage and contamination. To reduce these exposures, they are not physically handled. Handling is done using special tools, robots, and automation from component disk manufacturing through assembly of the disk stack on its spindle. Special containers are used to transport disks, and assembly operations use automation and robots to move and place disks during assembly.

Due to the close spacing between disks on the Disk Unit (#6100), special tools and techniques are used to prevent damage and contamination to disks or heads as they are merged together.

The actuator, a delicate assembly within the disk unit, requires special care and handling to protect its sensitive components. Heads and disks are the components most sensitive to damage, so unique head clips and head protectors are installed temporarily onto the actuator assembly. These clips help prevent damage from heads contacting each other or contacting a disk during merge or subsequent assembly. Trained technicians are equipped with grounding straps and special devices to prevent ESD damage and other damage to head suspensions, actuator voice coils, motor magnets, and electronic modules.

Contamination

Contamination control is critical when building a disk unit. Two types of contamination must be controlled: particulate contamination and magnetic particle contamination. Particulate contamination can cause undesirable head and disk interaction (head crashes). This contamination is minimized using an ultrasonic cleaning process and clean rooms for assembly. In addition, some parts have special plating or coatings to reduce exposure to flaking and corrosion (sources of particulate contamination).

Magnetic particle contamination causes degradation of signals. This source of contamination is normally associated with the rare-earth materials used to make voice coils and motors. These materials are coated to prevent

them from contributing to magnetic contamination within the process or during disk unit operation.

After components and assemblies are cleaned, subsequent assembly, and some testing, is conducted in clean rooms. These rooms are rated Class 100. This cleanliness is extremely important due to the close distance (approximately 305 nanometers) that the head flies above the magnetic media. Employees play a major role in maintaining a clean environment by wearing special hoods, gowns, and gloves to minimize contamination sources. All parts assembled in the clean room are controlled by an Automated Parts Handling System. The system monitors the inventory, automatically routes parts through the cleaners, and delivers parts at the request of operators in the clean room (see Figure 7).

Test Process

Testing is an integral part of the disk unit manufacturing process. Not only does it minimize costly rework by catching defects early in the process, but it ensures quality and reliability through statistical analysis and test process control.

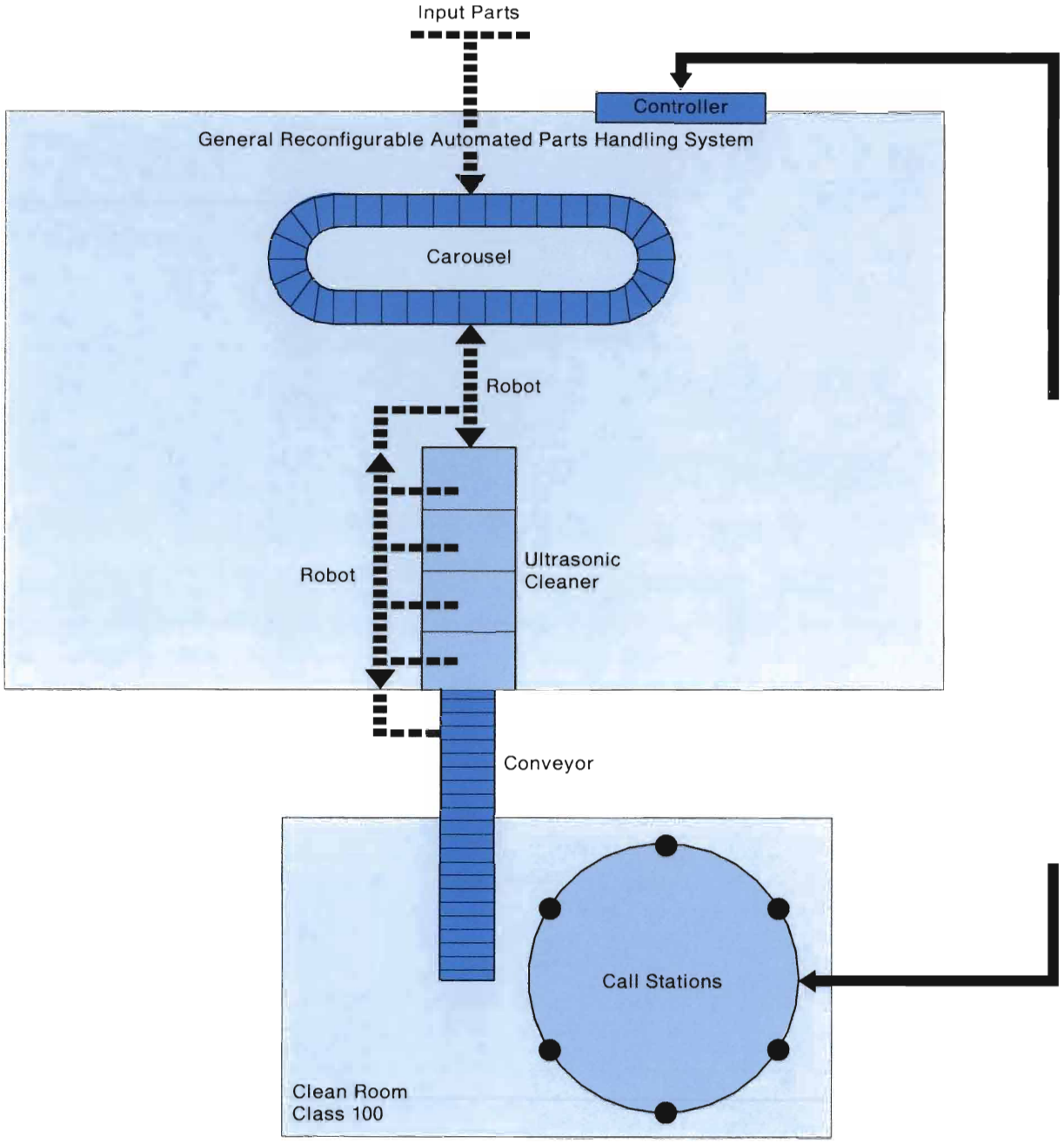


Figure 7 Clean Room Parts Delivery Control

Figure 8 8″ Device-Enclosure Tester

At the device-enclosure tester (see Figure 8), each file undergoes tests to verify that critical electrical and mechanical parameters are within limits. Actuator current, head-function switch time, and seek and settle times are measured under a variety of situations. Head-tangential, radial-offset, and actuator-bias current (current required to hold the actuator on track) are measured to ensure the mechanical system is operating correctly. A magnetic head read-and-write test is performed to measure error performance under forced off-track conditions. Start and stop times, motor start up, and constant speed idle current are also measured. A transfer function analysis test of the actuator control system is performed to detect mechanical vibrations that may affect disk unit performance. A particle-count test detects any contaminants left in the sealed enclosure.

To ensure data integrity, a surface analysis test is performed to locate media defect sites. If any sites are located, they are mapped by the disk unit's electronics and are not used for data storage. Test data is retrieved and sent to the manufacturing control system to be saved and to allow the disk unit to be routed.

The surface analysis method for the Disk Unit (#6100) uses a traditional analog test, where a constant frequency pattern is written on the disk and special detectors monitor the head signal for anomalies as it is read back from the disk. The test is designed to be fast and thorough.

Figure 9 8″ Device-Enclosure Undergoing Self Surface Analysis Test (SAT)

The surface analysis test method used on the 9332 Disk Unit is unique in that the unit actually tests itself. The disk unit is almost completely assembled in its enclosure when it undergoes surface analysis (see Figure 9). The microprocessors imbedded in the product electronics control the test so that no external equipment is needed.

Prior to shipment, the completed disk unit undergoes one final series of tests to detect any latent problems. All of the interface commands are processed and fault conditions are simulated. The disk unit is also run in a simulated operating environment, and then it is formatted for shipment.

Conclusions

Several new, unique techniques are used to assemble and test the Disk Unit (#6100) used in the 9404 System Unit and the 9332 Disk Unit. These activities, combined with defect-free, precision components and assemblies, allow the production and delivery of extremely reliable storage units for use in AS/400 systems and other computer systems.

Electronic Data Interchange

Describes the Electronic Data Interchange system and how it affects manufacturing not only in IBM, Rochester, MN, but throughout the entire IBM Corporation.

Richard E. Albrecht

Introduction
The goal to improve the quality of business communications between IBM and its suppliers, and improving productivity, reducing costs, and enhancing customer service, was met by installing a system using available state-of-the-art technology, thus producing a unique and innovative system with minimal invention.

IBM's implementation of the Electronic Data Interchange system is linked directly to the Professional Office System (PROFS) and integrated into the very fabric of the internal manufacturing applications, in a way transparent to the end user. The system architecture was designed to be used at all IBM manufacturing locations worldwide.

Adapting The Existing Network and Standards
The first objective of this project was to use nationally approved data communications standards. The American National Standards Institutes (ANSI) Accredited Standards Committee (X12) has been chartered to provide standard electronic data formats for generic business transactions, usable by any type of business entity (public, private, or governmental). These standard formats allow dissimilar computer hardware, with unique internal data file formats, to communicate electronically by converting them to a common format.

A second objective was to use an existing provider of networking services. The network that offered the necessary security was the IBM Information Network, providing proven secure-data transmission.

The IBM Information Network (located in Tampa, FL) is used to send notes and files, as well as business transactions including purchase orders, purchase-order acknowledgements, shipping schedules, shipment notifications, invoices, and payment notifications.

The network product, Information Exchange (a store-and-forward electronic mailbox system), acts as a buffer between IBM computer systems and those of the supplier. This removes any direct connection between the suppliers and IBM computers and allows communications to be restricted to designated agents at either end of the connection.

A third objective was to integrate the Electronic Data Interchange system into manufacturing and office systems without disturbing existing applications with which users were familiar and comfortable. In fact, existing internal applications are so interdependent that a change to one program could necessitate changes to many programs.

A modular design provided a solution that made it simple to add new business transactions to the list of transactions already exchanged electronically between IBM and its suppliers. Because of the way PROFS is linked to the Electronic Data Interchange system, the only difference between sending information within IBM or to an external supplier is the use of a different destination node and identifier with the system.

The fourth objective was to allow suppliers using the Electronic Data Interchange system to use their own hardware and software. The IBM Information Network allowed suppliers the flexibility of using IBM or other hardware in which they had already invested.

System Architecture
IBM must provide a single Electronic Data Interchange system solution to its suppliers, so that those doing business with multiple IBM manufacturing locations have an identical Electronic Data Interchange system interface. The architecture developed at IBM Rochester is the basis for the Electronic Data Interchange system architecture for the corporation. The four components are: the internal application base; the electronic-data standard conversion software; the send-and-receive software; and the IBM Information Network.

The internal applications appear unchanged and are accessed by an end user through a display station or an attached personal computer. A router acts as the system traffic cop, ensuring that transactions are routed correctly between the internal applications, the data conversion software, and the send-and-receive software. It consists of a series of programs that provide the bridge between each internal application and the Electronic Data Interchange system.

Conversion software maps inbound and outbound business transactions to the corresponding ANSI X12 transaction format. Notes and files that do not require conversion are passed directly to the send-and-receive software. (Although only ANSI X12 transactions are being used at this time, if a supplier has already used another electronic data format, the software could easily be converted to and from any nationally approved data standard.)

The IBM Information Network provides three types of send-and-receive software that allows any manufacturer's computer to connect to the Network. A Systems Network Architecture (SNA) connection allows IBM systems to link to the Network through a leased line or satellite connection using the SNA communications protocol. Remote job entry (RJE) allows an IBM mid-range system, or any computer not using SNA, access to the Network through a dial-up or leased line. Personal Computer Informational Exchange allows personal computers to use dial-up capability.

The Network provides connectivity, security, network management, maintenance, and billing, which simplifies the support required from IBM Rochester.

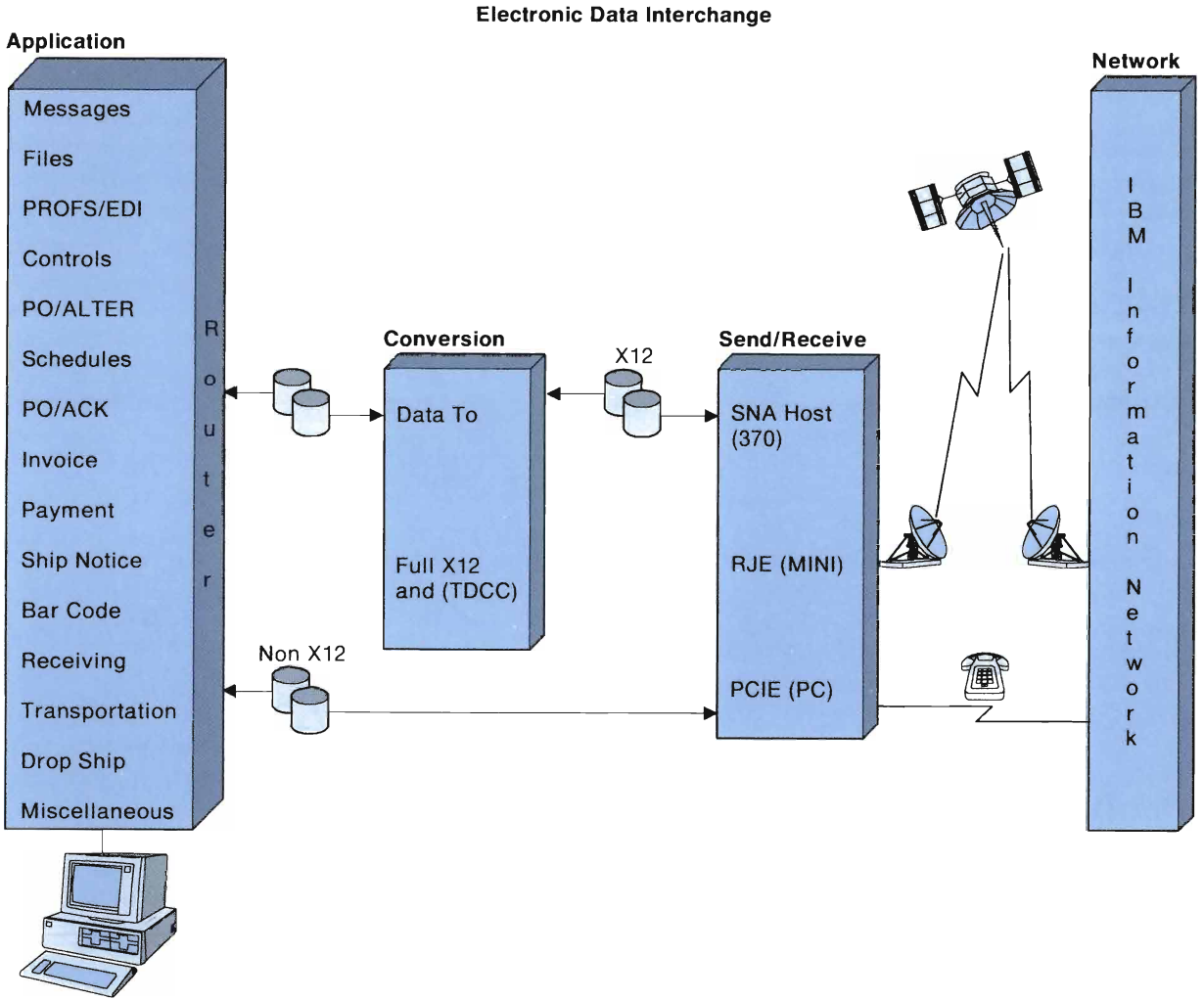
The supplier's architecture would contain the same elements as those at IBM. The supplier can start using a personal computer and a dial-up modem to exchange notes and files. The system can grow to take advantage of business transaction exchanges. The supplier can implement the transactions that would yield the greatest payback the fastest. Most industries start with either the purchase order or the invoice transactions, both of which are the basis for other transactions. Additional transactions can be selectively enabled as the supplier implements bridges to other internal systems.

Applications

All applications shown in Figure 1 are planned to be installed and available to our suppliers.

Two transactions that automate the receiving process internally and aid tracking of material shipments between IBM and the supplier are particularly important.

IBM's distribution-receiving function is automated based on an Electronic Data Interchange system interface coupled with the use of bar codes. Suppliers are asked to send an ANSI X12 shipping notice when material leaves their shipping dock. On this shipping notice, the predefined control number is converted to a bar code and affixed to the shipment. Suppliers can use preprinted bar



RSLL411-2

Figure 1 Electronic Data Interchange Manufacturing Systems Environment

code labels (with a unique set of control numbers) or print their own. When the shipment arrives at IBM, the bar code is scanned, and the system is automatically updated to reflect the receipt based on information preloaded on the system from the Electronic Data Interchange ship notice. This approach for receiving and moving material optimizes the IBM receiving effort and simplifies the supplier's tracking of shipments.

IBM is also pursuing Electronic Data Interchange connections with the carriers providing intercompany transportation of materials. The transportation companies will provide IBM and the supplier with status updates for tracking shipments. This is especially important in the continuous flow manufacturing world with daily, and even hourly, shipments.

Figure 2 summarizes the types of daily transactions that flow between buyer and seller. All of these transactions have corresponding electronic transaction formats.

Conclusions

The Electronic Data Interchange system provides the common solution for a common problem: business communications with suppliers. It provides the timely access and accurate information necessary to improve productivity, reduce costs, and enhance customer service, thus ensuring IBM's achievement of its continuous flow manufacturing goals.

Continuous flow manufacturing and the Electronic Data Interchange system are two of the tools that enable IBM to remain competitive in a highly competitive industry.

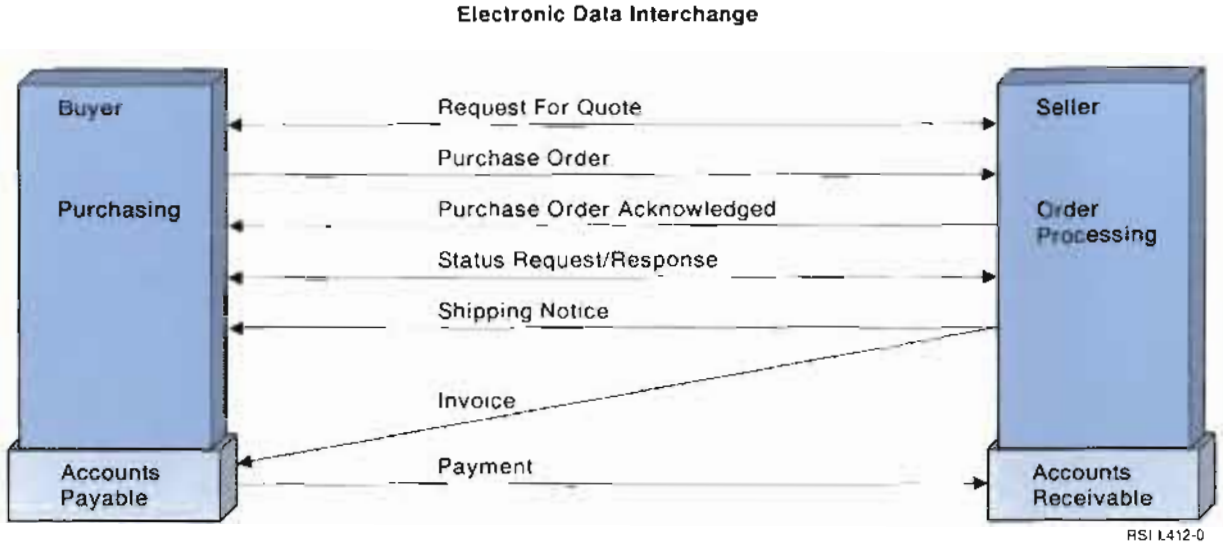


Figure 2 ANSI X12 Transactions

About the Authors

Richard E. Albrecht

Mr. Albrecht is a staff programmer in the Corporate Manufacturing Electronic Data Interchange project office.He joined IBM, Rochester, MN, in 1979 as a systems analyst in charge of internal purchasing and accounts payable applications. For the past two years, he has been working on the Electronic Data Interchange project with suppliers. He has coauthored a technical report on the Electronic Data Interchange system and has participated as a speaker in Electronic Data Interchange system seminars held by the IBM Information Network Marketing group. He received his BS degree in Computer Science and Finance from the University of Wisconsin at LaCrosse.

Mark J. Anderson

Mr. Anderson is an advisory programmer responsible for SQL, distributed data management, and data base as a member of the AS/400 design control group. He also represents the AS/400 system on the SQL control board. He has spent his career working on data base related architecture, design, and implementation. He joined IBM, Rochester, MN, in 1974 after receiving a BS in Mathematics from Luther College, Decorah, IA.

James H. Bainbridge III

Mr. Bainbridge is a senior associate programmer working in office/personal computer development. His experience has been largely in the development of host system-to-PC cooperative processing functions, including PC text assist and the PC Performance Monitor for the System/36. He received a BS in Computer Science in 1984 from the University of Wisconsin at LaCrosse.

Surinder P. Batra

Mr. Batra is a development programmer and manager in the communications development area. Mr. Batra defined the software structure and led the design and implementation of the IPCF architecture for the test system used in testing the I/O processors for the AS/400 family. In his most recent assignment, he has been responsible for the definition, design, and implementation of the microcode for the AS/400 Magnetic Storage Device Controller. Mr. Batra is a member of the Association for Computing Machinery. He received an MA in Mathematics from the University of Delhi, India; an MBA from McMaster University, Hamilton, ONT, Canada; and an MS in Computer Science from the University of Santa Clara, CA.

Donald L. Beck

Mr. Beck is a staff manufacturing engineer working in system stress testing. He joined IBM, Rochester, MN, in 1968 and has worked developing tester hardware and software within manufacturing in the areas of card test, hard disks, and subassembly test. He graduated in 1968 from the University of Nebraska at Lincoln with a BSME.

Neil C. Berglund

Mr. Berglund is a senior engineer in entry systems development. He holds 11 patents for processor and I/O controller work on System/3, System/38, and the AS/400 system. His most recent assignment has been in developing architectures for the AS/400 systems. Mr. Berglund holds a BSEE from the University of Minnesota at Minneapolis.

J. Howard Botterill

Mr. Botterill is a senior programmer in the software strategy, architecture, and planning group at IBM, Rochester, MN. He is responsible for the user interface strategy. He joined IBM Rochester in 1967 and helped develop the Multiple Terminal Monitor Task (MTMT) system for System/360. He worked on the Communication Control Program (CCP) for the System/3 and had the design control responsibility for the System/38 user interface. From 1982 to 1984, he worked at the System Products Division headquarters in White Plains, NY, coordinating the division's usability process. Since that time, he has worked on the design of the IBM Common User Access user interface and the design of the AS/400 interface. He received his BS in Mathematics from Wheaton College, Wheaton, IL, and his MS in Mathematics from the University of Michigan at Ann Arbor.

Daniel S. Brossoit

Mr. Brossoit, staff programmer, was the team leader of the location manager project on the AS/400 system. His previous assignments have included work on System/36 APPC, System/36 APPN, System/36 MLU, and PC Support/36. He joined IBM, Rochester, MN, in 1981 after receiving a BA in Quantitative Methods and Computer Science from the College of St. Thomas, St. Paul, MN.

Delbert R. Cecchi

Mr. Cecchi is an advisory engineer in the circuit technology group. Since 1973, he has worked on the design and application of VLSI in System/36, System/38, and the AS/400 system. He received BSEE and MSEE degrees from the University of Minnesota at Minneapolis.

Dennis A. Charland

Mr. Charland is an advisory information developer in the software strategy, architecture, and planning group, where he works on the strategy and design of user interface and user help facilities. He joined IBM, Rochester, MN, in 1977 and was involved in developing printed and online information for System/38 and System/36. He received a BA in Journalism from the University of Minnesota at Minneapolis in 1960. Prior to joining IBM, Mr. Charland worked at Univac, General Atomic, and the Aerospace Corporation.

Trent L. Clausen

Mr. Clausen is a staff engineer in the advanced systems engineering group. He was the microcode team leader for the asynchronous local work station controller. Previous assignments have included microcode design on the System/34 and System/36 local work station controllers. Mr. Clausen joined IBM in 1975 after receiving a BSEE from the University of Nebraska at Lincoln.

Richard L. Cole

Mr. Cole is a staff programmer with the data base management group. Previous experience includes working as a programmer and technical team leader on the data base, distributed data management, and query processing components of the System/38. He is currently responsible for the extended control program facility data base, query processing, journal management, and commitment control components of the AS/400 system. He is a member of the Association for Computing Machinery and received his BS in Computer Science from Michigan State University, East Lansing, MI.

Donald L. Conroy

Mr. Conroy is a new products administrator for AS/400 systems, responsible for the introduction of new system designs into manufacturing. He worked as a manufacturing engineer on the design of the AS/400 flexible manufacturing system and as a process quality engineer responsible for manufacturing process certification, manufacturing verification testing, and process capability studies. He holds an MS in Industrial Engineering and Operations Research from the University of Massachusetts at Amherst.

John T. Costello

Mr. Costello is a senior engineer working on advanced manufacturing technology. From 1984 to 1986, he worked on implementing continuous flow manufacturing activity for rigid-disk storage units and components. Since joining IBM in 1956 as an apprentice toolmaker, he has held staff and management positions in manufacturing, industrial engineering, and manufacturing engineering, including program manager in manufacturing technology planning. He has a BSME from the University of Minnesota at Minneapolis, an MS in Manufacturing Engineering from Boston University, Boston, MA, and an MBA from Mankato State University, Mankato, MN.

Earl A. Cunningham

Dr. Cunningham is a senior engineer in disk-storage recording component integration. From 1970 to 1974, he was Assistant Professor of Electrical Engineering at Lafayette College, Easton, PA. He joined IBM, Rochester, MN, in opto-electronics, originally working in optics and later in flex-file support. He moved to the fixed-disk drive mission when it began at IBM Rochester and has worked in that area since. Cunningham has five US patents presently issued, two US patents pending, and 22 disclosures published. He received his BS, MSEE, and PhD degrees from the University of Minnesota at Minneapolis.

Stephen J. Cyr

Mr. Cyr is a development programmer in performance evaluation, supporting AS/400 development. Previous assignments include applications support on System/370 and System/38, work on the System/36 5364 and the token-ring attachment, and manager of PC Support/3X development. He joined IBM Rochester, MN, in 1978 with BS degrees in Mathematics and Computer Science from Moorhead State University, Moorhead, MN.

Steven A. Dahl

Mr. Dahl is a senior programmer in the system design control group. Since joining IBM in 1970, his activities have included work on magnetic-ink character recognition and diskette I/O device support on System/360 and compiler applications, and operating system design for the System/3, System/32, System/34, and System/36 products. He is currently involved with general AS/400 design and architectural considerations, with primary emphasis on incorporating System/36 concepts and function into OS/400. Mr. Dahl received a BS in Mathematics and Computer Science from the University of Illinois at Urbana.

Gregory A. Dancker

Mr. Dancker is a staff engineer in the advanced systems engineering group. He was the hardware team leader for the synchronous local work station controller. Previous assignments have included hardware design on System/36 I/O controllers and adapters. Mr. Dancker joined IBM in 1978 after receiving a BSEE from the Milwaukee School of Engineering, Milwaukee, WI.

Michael J. Denney

Mr. Denney is a staff programmer working in performance comparisons. He has been involved in performance modeling and analysis since he joined IBM, Rochester, MN in 1982. He has a BS in Computer Science from Iowa State University, Ames, IA.

Carol A. Egan

Ms. Egan is currently a staff programmer in the communications group. She joined IBM in 1984 after working as a programmer/analyst for Deere & Co. She has worked on various communications projects on both the System/36 and the AS/400 system. She has a Comprehensive Mathematics and Computer Science BS from the University of Wisconsin at Platteville.

Earl W. Emerick

Mr. Emerick is an advisory programmer in the customer support design control group. He has held various assignments in software development on System/38, System/36, and the AS/400 system in RAS and customer support. His assignments on the AS/400 system included the development of the common I/O architecture and, key participation in problem management flows and structure. He also made significant contributions in the provision of the Technical Support and Information Access functions as a member of the design control group for customer support. He joined IBM, Rochester, MN after receiving his BS in Computer Science from Indiana Institute of Technology, Ft. Wayne, Indiana.

Wayne O. Evans

Mr. Evans is a senior programmer in the operational services group. He works on AS/400 programming, with overall design responsibility for security, work management, command language, messages, and interface. Since joining IBM in 1964, his experience includes computer monitoring of cardiovascular patients, commmunications I/O support for the System/3, and the Multiple Terminal Monitor Task (MTMT) terminal system for the System/360. Mr. Evans received a BS in Mathematics and Chemistry from Adams State College, Alamosa, CO, in 1962 and an MS in Mathematics from Kansas State University, Manhattan, KA, in 1969. Prior to joining IBM, he was employed by the NASA Lewis Research Center.

Ronald O. Fess

Mr. Fess is currently the manager of the system design group for the OS/400. He has previously worked on various development projects in System/38 CPF and VMC, OS/MFT, OS/MVT, OS/VS1, and OS/MVS. Mr. Fess is a member of the Association for Computing Machinery. Prior to joining IBM in 1969, he received a BS in Mathematics from Augustana College, Rock Island, IL, and an MS in Computer Science from the University of Iowa at Iowa City.

Eric L. Fosdick

Mr. Fosdick is an advisory engineer in the system design control group. He has participated in the design of System/3, System/32, System/38, and the AS/400 system in the areas of hardware, software, and architecture. He joined IBM, Rochester, MN in 1967 upon receiving a BSEE degree from Marquette University, Milwaukee, WI.

Mark R. Funk

Mr. Funk is a staff engineer in the microcode development organization supporting the AS/400 System Processor. His experience includes development of a System/38 work station controller, development of the 5550 Japanese work station in Japan, and the development of three IMPI processors. He holds three patents and has 14 published inventions. He received his BS in Physics from Northern Michigan University, Marquette, MI, and an MSEE from Michigan State University, East Lansing, MI.

Kevin P. Gibson

Mr. Gibson is a senior associate programmer involved in designing and developing the Magnetic Storage Device Controller microcode. His prior experience includes design and development of operating system and personal computer software. He is a member of IEEE and the Computer Society of the IEEE. Mr. Gibson received a BSEE in 1978 from the University of Wisconsin at Madison.

William J. Goetzinger

Mr. Goetzinger is an advisory engineer responsible for AS/400 system simulation. He was initially involved in the design and development of microprocessor-based line printer controllers for System/38. He later joined the System/38 processor development group as a hardware designer, which led to his involvement with processor simulation. Mr. Goetzinger joined IBM, Rochester, MN, in 1976 after receiving a BSEE from Iowa State University, Ames, IA. He has since received an MSEE degree from the University of Minnesota at Minneapolis.

William E. Hammer

Mr. Hammer is currently an advisory engineer in technology and processor development. Mr. Hammer worked on the early definition of the AS/400 bus and was responsible for the definition of the bus manager functions of the I/O bus. He has been involved in several products concerned with controlling I/O devices with microprocessors and defining the attachment of I/O devices to systems. He joined IBM in 1960 after graduating from the University of Illinois at Urbana with a BSEE.

Stephen P. Hank

Mr. Hank is an advisory engineer in the configuration development group. He joined IBM, San Jose, CA, in 1977 as a test engineer on the 3880 Device Controller. He transferred to IBM, Rochester, MN, in 1981 where he was assigned to engineering product support for the System/34 and, while serving in that capacity, developed the 62PC Data Recovery Program. Following that, he became the technical project leader for the disk adapter which attached the 9332 and 9335 Disk Units to the System/38, and most recently, the Multiple-Function I/O Processor. He received a BS in Engineering Technology from Southern Illinois University at Carbondale in 1977.

Barry W. Hansen

Mr. Hansen is a staff programmer working in the programmable work station group. He has been developing 5250 emulation products since 1983, and was technical team leader for work station function development of AS/400 PC Support. He holds BSEE and MSEE degrees from Washington State University, Pullman, WA.

Mark W. Hansen

Mr. Hansen is a senior associate engineer in the circuit package production center responsible for the design and build of functional card testers. He joined IBM, Rochester, MN, in 1983 and has worked on several different card testers during his career. He graduated from the University of Nebraska at Lincoln with a BSEE.

Raymond K. Harney

Mr. Harney is currently co-team leader of the APPN project. He has been involved with several products on the System/38 and the AS/400 system including APPC, SNA host connectivity, node type 2.1 connectivity, SNA management services, and I/O processor attachment on the AS/400 system. Mr. Harney received his BA in 1981 in Math, Physics, and Computer Science from Luther College, Decorah, IA.

John Y. Harrington

Mr. Harrington is an advisory information developer in the software strategy, architecture, and planning group working on user interface design and specifications. He previously worked in the information development group as an editor, where he worked on documentation for both System/36 and System/38. Prior to joining IBM in 1968, Mr. Harrington was employed by Univac as a publications editor. Mr. Harrington received a BA in Psychology/English in 1959 from the College of St. Thomas, St. Paul, MN.

Peter J. Heyrman

Mr. Heyrman is a staff programmer in the System/36 environment development group. Since joining IBM in 1981, his experience includes System/36 print spooling, System/36 command processing, and the System/36 environment. Mr. Heyrman received a BS in Computer Science from the University of Wisconsin at Oshkosh.

Merle E. Houdek

Mr. Houdek is currently a senior engineer engaged in performance analysis and modeling of processor and I/O hardware. His previous assignments have been with custom systems, the 3740 Data Entry system development group, and the System/38 development group. He joined IBM in 1964 after graduating with a BSEE from Tri-State University, Angola, IN.

Fred L. Huss

Mr. Huss is an advisory engineer in storage I/O subsystem development. He joined IBM in 1973 and initially worked on logic design for optical character recognition machines. He later became involved in microcode development for display stations and remote work station controllers, focusing on design and implementation of SNA and data stream protocols. He also was involved with early project management and system architecture definition for the AS/400 system. His most recent assignment has been lead microcode designer for the Magnetic Storage Device Controller. Mr. Huss received a BSEE in 1970 and an MSEE in 1973 from North Dakota State University, Fargo, ND.

David L. Johnston

Mr. Johnston is an advisory engineer in the customer support design control group. His assignments have been in hardware and software development and have centered around RAS engineering for several optical page products and the System/38. His assignments on the AS/400 system included the vital product data and system reference code specifications. He also managed the design of the tools set and processes used to produce printed copy and online reference-code translate tables. Mr. Johnston has seven US patents issued, one US patent pending, and eight disclosures published. He joined IBM, Rochester, MN, in 1963 after receiving his BSEE from the University of Minnesota at Minneapolis.

Christopher H. Jones

Mr. Jones is currently co-team leader of the APPN project. Previous assignments include work on System/36 APPC, System/36 APPN, node type 2.1 connectivity, SNA management services, and attachment of personal computers to the System/36 and the AS/400 system. Mr. Jones received his BS in Business from Rochester (NY) Institute of Technology in 1978 and his BS in Computer Science from DeVry Institute of Technology, Atlanta, GA in 1984.

Gary R. Karasiuk

Mr. Karasiuk is currently in the AS/400 architecture group working on the SAA application development environment issues. He has worked on several System/38 utilities, including SEU and CGU. He has also worked as a member of the System/38 design control group, representing utilities, languages, and office. He received his BS from the University of Manitoba at Winnipeg in 1981 and joined IBM, Toronto, ONT, Canada, as a programmer after graduation.

Gary L. Kearns

Mr. Kearns is an advisory engineer in system test engineering. Since starting with IBM in 1967, he has worked in new product quality engineering performing early entry support on card, optical character reader, and system products as well as in the component quality and reliability laboratory supporting vendor component evaluation and release. His recent assignments have been developing and promoting stress testing strategy and methodology appropriate for current and new systems products manufactured at IBM, Rochester, MN.

Harvey G. Kiel

Mr. Kiel is a staff engineer in the advanced systems engineering group. He was a microcode designer and microcode team leader for the synchronous local work station controller. Previous assignments include microcode design on the 5294 Remote Work Station Controller and the 5260 Retail system. Mr. Kiel joined IBM in 1978 after receiving a BSEE from South Dakota State University, Brookings, SD.

Harold F. Kossman

Mr. Kossman is a senior engineer in advanced systems engineering. His current interests include performance analysis and modeling of processor and I/O hardware, as well as system architectures and configurations. He has one patent and several published papers. Prior to joining IBM in 1978, Mr. Kossman was a member of the engineering staff at the 3M Corporation and at the Monsanto Corporation. He received his BSEE from the University of Missouri at Rolla in 1970.

Gary L. Landon

Mr. Landon is a development engineer and manager in intermediate storage development. He has worked in various areas of manufacturing engineering. In recent years, he has worked on developing manufacturing processes for storage products. This includes the 14-inch, 8-inch, and 5.25-inch disk units.

Charles A. Lemaire

Mr. Lemaire is an advisory engineer in the I/O processor architecture and design control group. His current responsibilities include I/O processor architecture and system performance analysis with emphasis on future enhancements. He joined IBM, Rochester, MN, in 1976 and has worked on System/38 HMC doing performance modeling and code optimization of queueing, task dispatcher, supervisor linkage, and virtual address translation microcode. He has also worked on the architecture of various processors. He holds one US patent, 10 technical disclosures, and is recognized for his performance contributions to the System/38 Model 7 Central Processing Unit. Mr. Lemaire received a BSEE in 1975 and has completed course work for an MSEE from the University of Minnesota at Minneapolis. He received an MBA from the College of St. Thomas, St. Paul, MN, in 1985.

Robert F. Lembach

Dr. Lembach is an advisory engineer in design systems. His current interests include optimization strategies for VLSI systems design encompassing timing, placement, and routing. Dr. Lembach received a BSEE from Marquette University, Milwaukee, WI, and an MSEE and PhD from Carnegie-Mellon University, Pittsburgh, PA. He joined IBM, Rochester, MN in 1979.

Richard J. Lindner

Mr. Lindner is an advisory programmer responsible for the software development process strategy. He joined IBM, Rochester, MN, in 1966 and participated in both engineering and programming development of System/3 with responsibility for developing and maintaining many aspects of the I/O supervisor. He then accepted an assignment in the development of the System/38 Control Program Facility. Since then, he has held management assignments in both development and development-support areas. He became a professional engineer through training in the IBM Undergraduate Engineering Education program in conjunction with the University of Minnesota.

Frank J. Lukes

Mr. Lukes is a development engineer and manager responsible for mechanical packaging and hardware integration of the 9404 System Unit. His technical assignments, prior to being appointed manager, were primarily in electronic packaging of work stations and related strategic technologies. He has held other management assignments in advanced technology packaging. Mr. Lukes joined IBM, Rochester, MN in 1970 after receiving a BSEE from the University of North Dakota at Grand Forks.

Gene A. Lushinsky

Mr. Lushinsky is an advisory engineer in storage I/O subsystem development. He joined IBM, San Jose, CA, in 1966, working in manufacturing and test engineering on disk storage devices. In 1971, he transferred to IBM, Rochester, MN to support the implementation and product development of

diskette devices. His responsibilities have included both hardware logic and microcode for small systems and I/O products. He has held microcode team leader positions and had architecture development responsibilities for work stations and storage products. His most recent assignments have been technical guidance for the implementation of the storage product architectures.

Robert W. Lytle

Mr. Lytle is a staff engineer responsible for the final manufacturing system test for the AS/400 system. He joined IBM, Rochester, MN in 1982 and has worked in both hardware and software development of manufacturing tests on system products. He received a BS in Mathematics in 1972 from South Dakota State University, Brookings, SD, and an MS in Engineering from the same institution in 1982.

Paul R. Mattson

Mr. Mattson is an advisory programmer in the communications area. His early communications assignments include working on the APPC I/O manager, 5250 display station pass-through, and the X.25 I/O manager. During those assignments, Mr. Mattson earned an MS degree in Computer Science from the University of Minnesota at Minneapolis. His most recent assignment included technical design leadership for the SNA-based data link control I/O managers on the AS/400 system. He is a member of the AS/400 communications design control group. He joined IBM, Rochester, MN in 1981 after earning BA degrees in Mathematics and Computer Science from Luther College, Decorah, IA.

James M. Mickelson

Mr. Mickelson is an advisory programmer in capacity planning tools development. He joined IBM in 1966 and worked on System/360 device support and compiler development until 1973. He then worked on System/3 communications support until joining the performance group in 1978. Work in this area has included performance modeling and capacity planning for IBM mid-range products. He received a BS in Mathematics from the University of Wisconsin at Eau Claire in 1966.

John A. Modry

Mr. Modry is an advisory programmer in the System/36 environment development group. He joined IBM, Rochester, MN in 1976 and worked on System/38 work management through 1981. Since then he has worked on System/38 3270 device emulation and had various development and architectural responsibilities in the development support area. Mr. Modry received a BS in Computer Engineering and an MS in Computer Science from the University of Illinois at Urbana.

James R. Morcomb

Mr. Morcomb is a senior engineer and manager of the customer support design control group. He joined IBM, Rochester, MN, in 1957 and has held various development and development management positions with a strong emphasis toward RAS, service, and customer support. He managed the engineering RAS development department on System/38 with primary responsibility for developing the advanced automated service support capability of the system. In 1982, he chaired the SPD task force that developed the mid-range systems RAS strategy which became the basis for the RAS support on the 9370 system and for system support. Since 1986, he has served as chairman of an interdivisional steering committee that oversees the worldwide implementation of system support.

Michael F. Moriarty

Mr. Moriarty is an advisory programmer in the software strategy, architecture and planning group. He joined IBM, Rochester, MN, in 1968. From 1969 to 1971, he was in the US Navy as a programmer at the Bureau of Naval Personnel (BUPERS), Washington, D.C. He has worked on the software development of the System/3, System/32, System/34, System/36, and the AS/400 system. He received a BA in Mathematics from the University of Missouri at St. Louis in 1968.

Timothy J. Mullins

Mr. Mullins is an advisory engineer involved in processor performance analysis. He has done work in the design and development of I/O controllers for System/38 user display devices. Mr. Mullins later became involved in processor development in logic design and timing analysis. He joined IBM, Rochester, MN, after receiving his BSEE degree from the University of California at Berkeley in 1977. In 1982, he received his MSEE degree from the University of Minnesota at Minneapolis.

Hjalmar H. Ottesen

Dr. Ottesen, a Senior Engineer, joined IBM in 1962. He is currently working in advanced rigid-disk servo technology and applications. He has held various technical positions in advanced development of magnetic recording channels and position servo control systems for tape, disk, and mass-storage devices. He also spent four years in an IBM World Trade branch office working with customers on APL programming applications. He has nine US patents issued and 16 disclosures published. He received his BS, MS, and PhD degrees from University of Colorado at Boulder in 1961, 1962, and 1968, respectively.

Renato J. Recio

Mr. Recio is a project engineering manager responsible for the software and hardware architecture and design of I/O adapters and processors. He joined IBM, Rochester, MN in 1982 as an engineer working on I/O device attachments for the System/36. He holds several disclosures relating to storage I/O adapters, processors, and devices. He is currently working on data structures for storage I/O devices. Mr. Recio has a BS in Electromagnetics/Electronics from the University of Illinois at Chicago and is taking coursework towards an MBA from the University of Minnesota at Minneapolis.

Arthur P. Reckinger

Mr. Reckinger is a senior engineer and manager in systems packaging. He has had assignments in card machine technology, key entry technology, electronic packaging technology, and systems architecture. Other assignments have included product development on the 3747 and systems development on the System/38. He joined IBM in 1967 after earning his BSEE and MSEE from the University of Missouri at Rolla.

Kenneth R. Reid

Mr. Reid is a senior engineer and manager of high performance systems hardware product design. His experience includes design and development of I/O (card readers, card punches, optical readers, printers, and check readers), System/38 service processor design, and System/38 RAS. For the past several years, he has managed several different departments in System/38 and System/36 development, mainly in the area of diagnostics and microcode development. He joined IBM in 1964 after receiving his BSME and MSME from North Dakota State University, Fargo, ND.

Jeffrey E. Remfert

Mr. Remfert is an advisory engineer in the advanced systems engineering group. He was the microcode team leader for the synchronous local work station controller. Previous assignments have included both hardware and microcode design on System/34 and System/36 I/O adapters. Mr. Remfert joined IBM in 1970 after receiving a BSEE from the University of North Dakota at Grand Forks.

James R. Rubish

Mr. Rubish is a senior associate engineer in advanced systems engineering. His primary responsibilities included the design, implementation, and verification of the AS/400 processor. Mr. Rubish previously participated in processor design and support for System/36. He joined IBM, Rochester, MN, in 1984 after receiving a BSEE from North Dakota State University, Fargo, ND.

Larry F. Saunders

Mr. Saunders is an advisory engineer in automation technology. Since joining IBM, Rochester, MN, in 1976, he has participated in the development of the System/32 and System/34 processor hardware and provided support and education to other engineers using simulation, LSSD test generation, and structure-processing design automation tools. Since 1982, he has led the development of new design automation tools related to hardware description languages and high-level logic synthesis. Mr. Saunders is a member of the Computer Society of the IEEE. He received a BSEE degree in Computer Hardware Design from the University of Illinois at Chicago.

Quentin G. Schmierer

Mr. Schmierer is an advisory engineer in the central processor development group. His experience at IBM has included developing a System/38 flexible disk controller, a high-speed tape drive controller, and development work on three IMPI processors. Mr. Schmierer holds an IBM First-Level Invention Award in recognition of a patent application and nine published inventions. He received a BSEE in 1976 from North Dakota State University, Fargo, ND.

Michael J. Snyder

Mr. Snyder is an advisory programmer in the system design control group. He joined IBM, Rochester, MN, in 1978 and has held various assignments in software development and management on System/38 and the AS/400 system. His AS/400 assignments included the initial design for the customer support functions as part of the customer support design control group. Mr. Snyder received a BS in Mathematics in 1970 and an MS in Computer Science in 1976 from the University of Missouri at Columbia. Prior to joining IBM, he was employed by Texas Instruments and also served in the US Navy.

Duane A. Spencer

Mr. Spencer is an advisory planner responsible for system hardware quality and reliability. He joined IBM as a customer engineer in Hammond, IN, in 1967. In 1977, he transferred to IBM, Rochester, MN, as a member of the NSD service planning staff. Following various systems support and development assignments in service planning, he joined the advanced systems development group in 1984.

Zanti D. Squillace

Mr. Squillace is an advisory engineer in system mechanical development. He joined IBM in 1962 and has had many technical assignments in optical character recognition and systems development. He received a BSME from the University of Minnesota at Minneapolis and is a registered professional engineer in Minnesota.

James C. Stewart

Mr. Stewart is a staff programmer in performance measurement and analysis. Past experience includes work in the US Navy, as an instructor in their Nuclear Propulsion Training program, and work for Sperry Univac Corporation, in the area of computer-assisted instruction. After joining IBM in 1977, he worked in a variety of assignments, all associated with the System/38. For the past five years, he has been involved in performance measurement and analysis work on the System/38. He received his BS in Mathematics from Moorhead State University at Moorhead, MN, in 1967.

Richard A. Tenley

Mr. Tenley joined IBM in 1965 in Kingston, NY. He had various power supply design responsibilities which included the TSR-6 family used in the System/370 158 and 168. In Rochester, he has held several assignments in power systems development. Mr. Tenley received a BSEE from the University of Minnesota at Minneapolis in 1960.

Dale J. Thomforde

Mr. Thomforde is an advisory engineer in the central processor development group for the AS/400 system. He was heavily involved in the performance aspects of the processor design. Mr. Thomforde has one US patent pending and 13 disclosures published. He received a BSEE in 1973 from the University of North Dakota at Grand Forks.

Keith L. Thompson

Mr. Thompson is an advisory engineer responsible for system hardware quality and reliability. Past IBM assignments have included the development of card I/O products; display and work station systems; display station, printer, and disk unit system adapters; and memory subsystems. He joined the advanced systems development group in early 1984. Mr. Thompson joined IBM, Rochester, MN, in 1965 after he received BSEE and MSEE degrees from North Dakota State University, Fargo, ND.

William A. Thompson

Mr. Thompson is an advisory programmer in advanced systems engineering. He is lead designer for the AS/400 Service Processor microcode. Before joining advanced systems engineering, he had a variety of assignments within the System/38 programming center. He holds an invention disclosure for high-level data addressability. He graduated from State University of New York at Cortland with a BS in Secondary-Education Mathematics and received an MS in Computer Science (Operating Systems) from Rensselaer Polytechnic Institute, Troy, NY.

John N. Tietjen

Mr. Tietjen is an advisory engineer in AS/400 engineering development. Past assignments include printer adapter hardware and microcode, work station controller microcode, and department manager. Mr. Tietjen has held various I/O related assignments in System/3, System/38, and AS/400 engineering. He joined IBM, Rochester, MN, in 1970 after receiving his BSEE degree from Arizona State University, Tempe, AZ.

C. David Truxal

Mr. Truxal is a senior programmer and manager of the design control group for AS/400 Office. He joined IBM in 1967 in Kingston, NY, to work on virtual storage management. After serving in the US Navy, he worked for Control Data Corporation. He rejoined IBM, Rochester, MN, in 1977. His assignment was in the System/38 Architecture and Design Control groups, working on the system's integrated data base and security design. From operating system design responsibilities, Mr. Truxal moved into management of the high-level languages and utilities design control group for System/38 and, from there, to an office planning position for System/38.

Thomas M. Walker

Mr. Walker, a staff engineer, has held a variety of assignments within processor development. He holds one US patent and two invention disclosures in system performance measurements. He graduated from the University of Washington at Seattle with a BSEE.

James O. Walts

Mr. Walts joined IBM, Poughkeepsie, NY, in 1968 as a programmer. Since then, he has held technical and managerial positions in communications-oriented advanced technology projects. In addition to participating in the early design efforts of System/38 and the AS/400 system, he was a lead designer of the LU type 6.2 and SNA implementations. He is currently the manager of AS/400 Strategic Communications and Networking. Mr. Walts received his BA in Secondary-Education Mathematics from the State University of New York at Potsdam and his MS in Applied Science Systems Programming and Languages from the College of William and Mary, Williamsburg, VA.

Thomas J. Warne

Mr. Warne is a staff engineer in rigid-disk storage unit development. He started with IBM in 1977 as a test equipment design engineer in disk manufacturing. He was part of the original team that produced IBM Rochester's first 8-inch disks. Since 1982, he has worked in test engineering on surface analysis test equipment and was a member of the team that developed the self-test approach for surface analysis on the 8-inch 9332 Disk Unit. He has a BSEE from the University of Wisconsin at Madison.

David G. Wenz

Mr. Wenz is a senior programmer in the office design control group. He started his career on compilers, and then moved into end-user utilities, such as source entry, data entry, screen design aid, and text editors. He has been involved in office products for several years, including DisplayWrite/36, shared folders, and the PC Support Organizer menu. He has three US patents pending and 44 disclosures published. He graduated in 1967 with a BA in Mathematics and Psychology from the University of Wisconsin at Oshkosh.

David N. Youngers

Mr. Youngers is an advisory programmer working in office/personal computer development. He is recognized for his work with System/38 Source Entry Utility, Screen Design Aid, and Text Management, and with System/36 DisplayWrite/36 and PC Support/36 Office products. He joined IBM, Rochester, MN, in 1979 after graduating from Iowa State University, Ames, IA, with a BS in Computer Science.