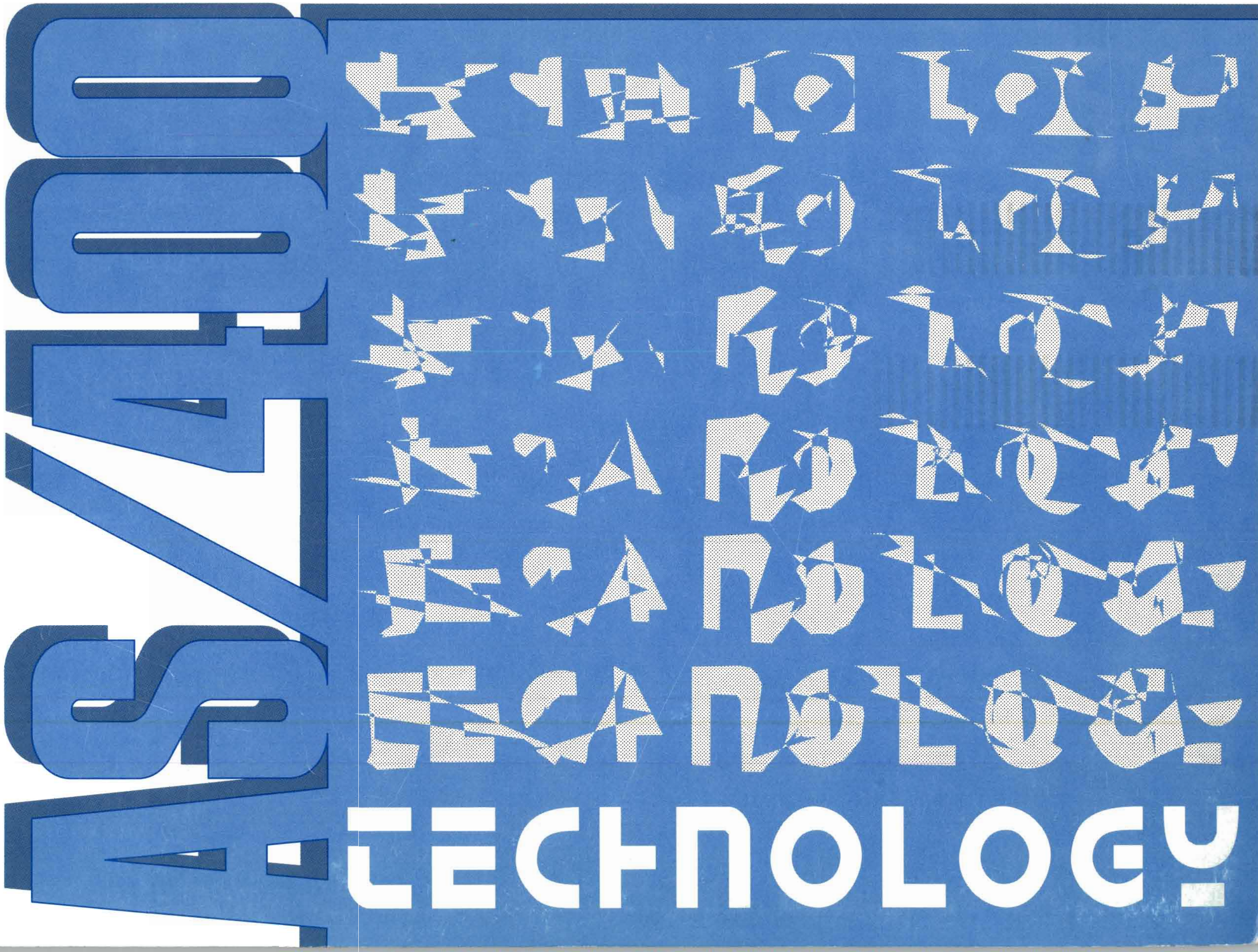
IBM Application System/400™ Technology Journal

Version 2



IBM

IBM Application System/400™ Technology Journal

Version 2

Malcolm Baldrige National Quality Award

IBM Rochester 1990 Winner

The 1990 Malcolm Baldrige National Quality Award was received by the IBM Rochester team, who develops and produces the AS/400 system. This award was established in 1987 by Congress and is presented annually to companies that exhibit excellence in all aspects of quality.

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page iv.

First Edition (January 1992)

International Business Machines Corporation provides this publication "as is", without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The articles will not be updated to incorporate design changes or new developments.

Order publications through your IBM representative or the IBM branch serving your locality. Publications are not stocked at the address given below.

Address comments concerning the content of this publication to:

Attn Department 245
IBM Corporation
3605 Highway 52 N
Rochester, MN 55901-7899 USA

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you or restricting your use of it.

© **Copyright International Business Machines Corporation 1992. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Foreword

Version 1 of the AS/400 system was introduced in June of 1988 and has been accepted enthusiastically by customers worldwide. Since then it has been enhanced with a number of releases, which added new levels of performance, usability, and function.

Version 2, which was announced in April 1991, marks a new milestone in AS/400 response to continuing customer requirements. It represents a significant enhancement on Version 1 and concentrates on a number of focus areas.

Version 2 provides extensive new functions and enhancements, particularly in the area of communications and network connectivity. These complement the software advances made in cooperative processing, distributed relational database, and usability.

The design team extended the advanced architecture of the AS/400 system with the implementation of the first *n*-way processors for the system, which is the approach used to achieve the high levels of performance now offered at the top of the range. At the low end, the introduction of new models makes the advanced functions of the AS/400 system available to an ever-widening spectrum of users. The effect is to make the more advanced applications, such as telephony, facsimile, image, and knowledge-based solutions, far more affordable and bring them within the reach of more and more customers.

It is the implementation of leading edge technology, such as IBM's latest CMOS chips containing up to 75,000 logic circuits on each 12.7-mm chip and the new 64-megabyte intelligent main storage card on Models D50 to D80, that makes these advances possible. On these models new laser-driven fiber-optic buses increase the speed at which data can be transferred to input/output devices and permit them to be located away from the processor.

Throughout the manufacturing process, teams have made changes to manufacturing, packaging, and testing processes to reduce the production cycle time while improving product quality. These and other innovative technologies introduced through the new AS/400 models enable us to deliver improved price/performance.

IBM is committing to market-driven openness by making the integrated system functions available through architected application programming interfaces while preserving AS/400 traditional strengths. The implementation of open systems interconnection and IBM's stated intention to embrace components of the industry standards of POSIX and OSF/DCE form key parts of the AS/400 openness strategy.

The delivery of a completely new series of compatible systems in April of 1991 was not a trivial task. It involved the work of many dedicated professionals in resolving the programming, engineering, and manufacturing challenges throughout the development cycle. This publication is a collection of articles on the design and development of some of the functions and licensed programs that are part of Version 2.

These articles describe some of the innovation, technology, and design that continue to make the AS/400 system a leading platform in providing advanced business solutions and improving our customers' competitive edge.

David L. Schleicher
ABS Director of Development
IBM Application Business Systems
Rochester, Minnesota

Notices

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 product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

This publication could contain technical inaccuracies or typographical errors.

This publication may refer to products that are announced but not currently available in your country. This publication may also refer to products that have not been announced in your country. IBM makes no commitment to make available any unannounced products referred to herein. The final decision to announce any product is based on IBM's business and technical judgment.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This publication contains small programs that are furnished by IBM as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. All programs contained herein are provided to you "AS IS". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*), used in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

Application System/400
AS/400
CallPath
Common User Access
CUA
Distributed Relational Database Architecture
DRDA
IBM
NetView
OfficeVision
Operating System/2
Operating System/400
Operational Assistant
OS/2
OS/400
Personal System/2
Presentation Manager
PS/2
QMF
RISC System/6000
SAA
SQL/400
Structured Query Language/400
System/370
System/390
Systems Application Architecture
SystemView
400

The following terms, denoted by a double asterisk (**), used in this publication, are trademarks of other companies as follows:

High Density Plus	Teradyne, Inc.
Hicom	Siemens Aktiengesellschaft
Intel	Intel Corporation
IRX9000	Teleos Communications, Inc.
Microsoft	Microsoft Corp.
Meridian SL	Northern Telecom Limited
Meridian 1	Northern Telecom Limited
Northern Telecom NT	Northern Telecom Limited
Novell	Novell, Inc.
Rolm	Siemens
RUMBA	Wall Data Incorporated
Siemens	Siemens Aktiengesellschaft
Teleos	Teleos Communications, Inc.
UNIX	UNIX System Laboratories, Inc.

Preface and Acknowledgments

The articles in this publication, about new and different methods used in designing and developing Version 2 of the AS/400 system, were written by and for technical professionals. The articles are not intended to describe product features or use, nor is the book intended to replace IBM reference manuals or user guides.

All the authors were volunteers, and their significant time and effort, on top of already very busy schedules, are greatly appreciated.

That this Version 2 edition of *Application System/400 Technology* even came about is due to the vision of Dick Odell, who championed it and won its “commission”; and to the leadership of Darrell Horn, who recruited an enthusiastic team blending technical, editorial, and administrative skills, then “owned” and supported the book throughout its development process.

The technical coordinators for the sections were Frank Soltis for Programming, Merle Houdek for Engineering, and Joe Rommel for Manufacturing. They ably identified prospective authors, ensured balanced coverage of topics, coordinated and communicated process details–and still maintained a sense of humor throughout!

Trying to maintain some consistency and conformance to guidelines on a project involving 89 authors and 33 topics is no small task, and the editorial/graphics team richly deserves recognition: Cheryl Henning, the editor, who worked diligently and skillfully to maintain the book's integrity across many different writing styles and levels of detail; Michelle Meyers, the editorial assistant, who cheerfully and deftly coped with new coders, new process, new format, and new challenges; Bob Harms and Syl Gerdts, the artists, who creatively and productively developed (and revised, and revised again) the illustrations for the articles as well as the cover and section divider designs.

Thanks to all of you for making this another high-quality Rochester product, and for making my job such a delight. And thanks to my management, Mike Ransom and Kevin Moritz, for allowing me to take this very exciting and challenging assignment.

I hope you readers enjoy reading this book as much as we enjoyed producing it.

Sylvie Nickel

Sylvie Nickel
Managing Editor

Contents

Figures

AS/400 Version 2 Overview

Provides an overview of the AS/400 Version 2 system and introduces the topics covered in this technology journal.*

Frank G. Soltis, Merle E. Houdek, and
Joseph T. Rommel

Introduction

The system overview article of the original *IBM* Application System/400* Technology* journal, SA21-9540, described the AS/400 system as "a broad new family of general purpose mid-range systems." It continued to state that the "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."

Since the introduction of the AS/400 system and the publication of the original technology journal in 1988, much has changed. The underlying architecture, however, is still the same. The basic system concepts of layered machine architecture, object orientation, and single-level addressability are fundamentally unchanged as is the concept of a single operating system that provides complete end-user and application portability across all models.

The addition of new hardware, software, and manufacturing technologies has greatly enhanced the capabilities and characteristics of the AS/400 system. The introduction of the Version 2 AS/400 system provides an appropriate time to produce a new technology journal to highlight these many new technologies that have been incorporated since 1988.

Like the original technology journal, this publication contains a collection of articles on the design and development of the AS/400 system. The articles are organized into three parts: Programming, Engineering, and Manufacturing. Written by technical and professional people who work in these areas, these articles describe some of the innovation, technology, and design built into the AS/400 system.

Programming Overview

The Version 2 software of the AS/400 system incorporates numerous new technologies as well as enhancements to the technologies that were originally introduced with the system. Figure 1 on page 3 shows a general software structure for Version 2 of the AS/400 system, highlighting those areas covered by the programming articles. This structure is built on the same advanced, extendable system architecture that was introduced with the AS/400 system. The result is to preserve customer investment in education and application programs while providing significantly new and enhanced functions.

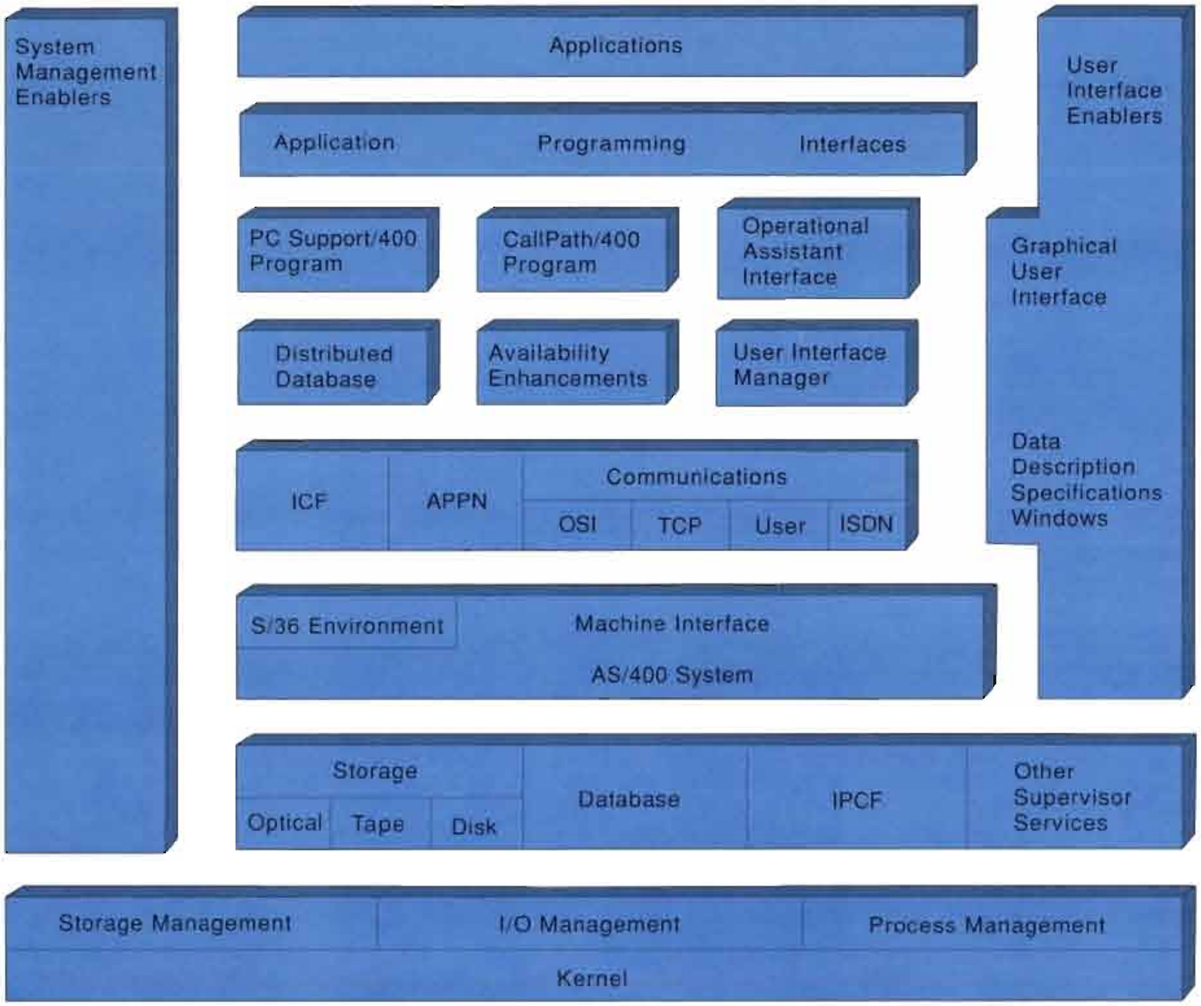
Fifteen articles have been selected to describe some of the software technologies of Version 2. The articles are divided into five sections: User Interface, Application Software, Cooperative Processing Support, Communications Support, and System Support.

User Interface

At Version 1, the AS/400 system introduced a new dimension in user interface consistency and advancements that increased ease of use and simplified work activities. The user interface was simple and self-guiding for new users, while still being efficient and productive for professional data processing users.

This emphasis on the user interface continues in Version 2. The four articles in this section describe the major advances since the AS/400 system was first introduced.

The Operational Assistant* user interface is new to the Operating System/400* (OS/400*) licensed program. It simplifies common system tasks while still providing full access to the power of the AS/400 system. The primary motivations for providing Operational Assistant are to make end users more productive, to free up the technical support personnel, and to minimize the amount of training needed to use the AS/400 system. The article by Ransom, Schmidt, and Massaro describes the Operational Assistant interface and the benefits that are achieved.



APPN	=	Advanced Peer-to-Peer Networking
ICF	=	Intersystem Communications Function
IPCF	=	Interprocess Communications Function
ISDN	=	Integrated Services Digital Network
OSI	=	Open Systems Interconnection
TCP	=	Transmission Control Protocol

TECH126-2

Figure 1. *AS/400 Software Structure*

Another major increase in user productivity can be accomplished through the use of a graphical user interface. Cooperative graphical user interface applications that combine the strengths of both the AS/400 system and a programmable work station can be built with relative ease. The article by Duffield, Eikenhorst, and Richards discusses the models of implementation and the technologies behind a graphical user interface on the AS/400 system.

At the introduction of the AS/400 system the user interface to the OS/400 operating system was provided by an internal component called the user interface manager. The help function for the user interface manager is now available to application developers to provide help information from application panels and from command prompting panels. Allen's article describes this new alternative for providing online application help.

Windows have been incorporated into AS/400 application programs for some time to improve the usability of the applications. Prior to Version 2, creating windows was not an easy task. Ulwelling describes the additions to data description specifications that simplify window creation and give additional window capabilities.

Application Software

The AS/400 system is first and foremost an application system. Advanced technology applications play an important role in the success of this system. The ability to incorporate image, voice, telephony, and artificial intelligence technologies into applications is a hallmark of the AS/400 system.

Underlying much of the advanced application support is the integrated, relational database. The database facilities in the AS/400 system continue to be enhanced with every new release. Nowhere have these enhancements been more significant than in the incorporation of distributed relational database technology.

The two articles selected for this section highlight one of the advanced technology application interfaces and the distributed relational database technology.

The choice of which advanced technology application interface to select for this journal is difficult because all of them are important. The desire of customers to integrate voice and data into their application programs makes the CallPath/400* licensed program a logical choice. The article by Bruner describes this licensed program, which provides an application programming interface to integrate functions and information from a variety of telephone switches into AS/400 applications.

The distributed relational database article by Broich, Egan, Tenner, Ramler, Wulf, and Kan describes the technologies used in the AS/400 distributed database manager and explains how they allow users to access data on remote systems. It also describes how to create a distributed application program.

Cooperative Processing Support

Cooperative processing continues to be an important focus for the AS/400 system in Version 2. Cooperative processing is the name used to describe the combining of the personal computer and AS/400 system environments to support advanced functions and applications. The ability to divide the function of a program between the personal computer and the host system was first introduced on the System/36. Similar support was incorporated on the AS/400 system at its introduction in 1988. Since then, numerous enhancements have been added. The two articles in this section describe how personal computers and AS/400 systems can work together to run application programs.

The first article by Wall, Schloss, Krueger, Scholten, Priniski, and Cook explains in more detail the concepts behind cooperative processing and how the AS/400 system supports these concepts. The PC Support functions that support a cooperative environment are also described.

Many functions in PC Support/400 have been added or enhanced since it was introduced in Version 1. The second article in this section by Wall, Wenzel, Kramer, Glowacki, Krueger, and Bukowski describes the changes and additions that have been made.

Communications Support

When the AS/400 system was first introduced, its communications and networking structure was described as one that would meet the data communications functions of the day as well as lay the foundation to meet future requirements. In Version 1 of the OS/400 licensed program, the advanced networking functions, known as advanced peer-to-peer networking (APPN), were incorporated. APPN was first introduced in the System/36 and System/38 to accomplish system-to-system and program-to-program communications. Transmission Control Protocol/Internet Protocol (TCP/IP) was also recognized as an important first step in fulfilling customer requirements for interoperability in a mixed vendor environment and was introduced in Version 1. Both TCP/IP and APPN have been enhanced for Version 2.

The commitment to open systems continues in Version 2 with the introduction of open systems interconnection (OSI) support and AS/400 user-defined communications support. Finally, the emerging new technology in the field of data and voice communications, integrated services digital network (ISDN), is supported in Version 2. The five articles in this section describe these communications technologies.

APPN has had several enhancements in the areas of usability, availability, connectivity, and performance since it was first introduced on the AS/400 system. The article written by Christenson, Cossack, Frett, and McGinn describes the details of these enhancements.

Kramer, Shih, and Cook's article explains the OSI reference model and describes the incorporation of this technology into the AS/400 system. The programming interface used by customer application programs is also discussed.

The AS/400 user-defined communications support, which provides an interface to the system communications functions, is presented in the article by Jones. An example showing how the application interface can be used to include the AS/400 system in a heterogeneous environment is also given and discussed.

The digital highway known as ISDN is the subject of the article by Carlson, Herring, and Jongekryg. The article describes the Version 2 support that provides for the direct attachment to an ISDN through an integrated adapter.

The collection of vendor-independent communications protocols that support peer-to-peer connectivity functions, known as TCP/IP, is explained in the article by Dick, Beierle, and Bauman. The three common application protocols used by most TCP/IP implementations are also described.

System Support

The topic of system support covers a wide range of subjects from online education to security, from automated problem reporting to support for multiple national languages. The two specific subjects selected for this section deal with topics of great interest to AS/400 customers today: systems management and system availability.

Systems management functions help a customer manage AS/400 systems in either standalone or network environments. As more and more AS/400 systems are installed in complex networks, the need for central enterprise management is increasingly important. Version 1 built the strategic architectural foundation for systems management functions on the AS/400 system. With that foundation, Version 2 expands the functions and capabilities. The article by Emerick and Morcomb describes the base established in Version 1 and the enhancements offered by Version 2.

The second article deals with availability and recovery functions on the AS/400 system. These topics take on increased interest as customer applications and systems become more mission critical. Crowley, Kelle, Martin, and McDermott describe the variety of availability offerings now offered on the AS/400 system. This article provides a detailed description of the theory and operation of these availability offerings.

Engineering Overview

The Version 2 hardware architecture of the AS/400 system continues to complement 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 prior IBM systems.

Typically, processors from the high and middle part of a product line of older versions are used in the middle and low part of the product line of newer versions. Thus, reuse of the older designs make up the low to middle part of the product line while new technological advances are used in the high-end of the product line. For this reason most of the engineering articles are about technology in the high-end of the AS/400 product line.

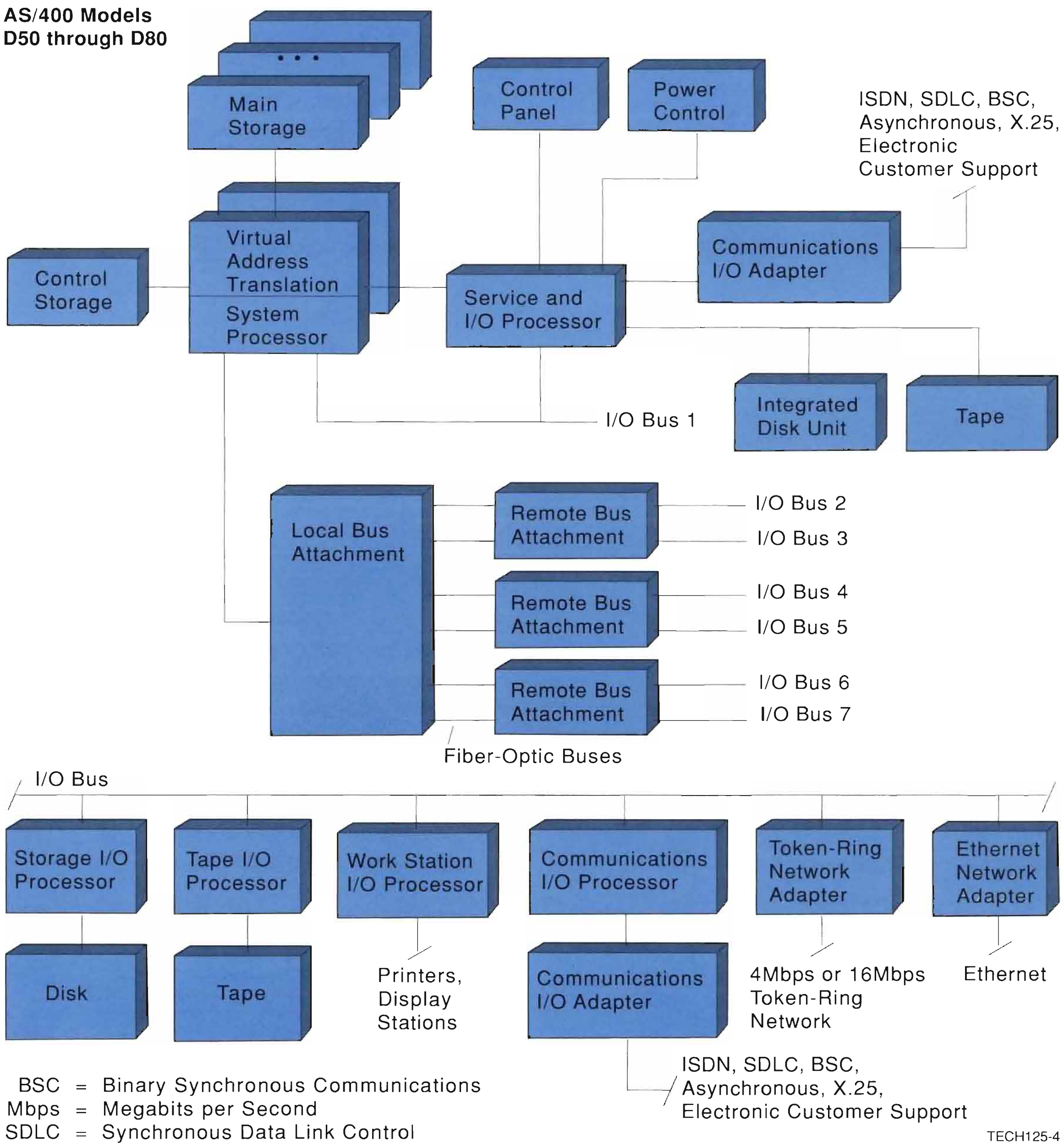
Processors and Main Storage

Figure 2 on page 6 depicts the hardware structure for the Version 2 D50-D80 models. These models use the latest technology improvements including a two-way processor in the D80 model. The processor architecture provides a single design that supports a single processor and a two-way processor with minimal changes to the software. This increases the computing power to two and one-half times the power of the Version 1

AS/400 system while keeping the software interface consistent between a single processor and the two-way processor. The article by Bahr, Levenstein, McMahon, Mullins, and Wottreng describes the multiprocessor architecture and design.

Main storage uses 4Mb (Mb equals 1,024 bits) main storage chips. Moving functions normally done in the system processor to the main storage cards is a critical component to *n*-way multiprocessing architecture. Eikill and Jensen describe the design and technology of the main storage. Two other articles covering the processor technology are authored by Cox, Johnson, Rudolph, and Williams and by Douskey and Kaliszewski.

Sophisticated development tools are used to design and test the logic and performance of the system processor chips and cards. Two articles, one by Schmierer and Wottreng and the other by Lembach, Borkenhagen, Elliott, and Schmidt, describe the development tools and processes used in developing the Version 2 processors.



BSC = Binary Synchronous Communications
Mbps = Megabits per Second
SDLC = Synchronous Data Link Control

Figure 2. *AS/400 Models D50-D80*

Input/Output

Version 2 hardware uses fiber optics to attach input/output (I/O) buses to increase I/O bandwidth and distance. I/O processors can be placed a greater distance from the system processor providing a greater flexibility in configuring systems. Fiber optics also improves the noise immunity in these systems, thus improving the availability of the system to the customer. Beukema, Block, Karst, Soderstrom, and Siljenberg describe the technology in the fiber-optic I/O bus.

The integrated disk unit provides for greater availability. In the event of a long utility power outage, the operating system saves the contents of main storage to a space on the integrated disk unit. Batteries keep the necessary components active to allow this to happen. This capability provides for quicker recovery time when utility power is restored. The direct attachment of the integrated disk unit to the service processor allows an initial program load of the service processor and the system processor. Failures can now be isolated to the I/O bus, I/O processors, or the system's I/O devices. The capability to provide better error isolation improves the ability to identify failed units, which improves the availability.

New I/O processors are included in Version 2 and provide better communications capabilit; and performance. An article by Kiel and Peterson describes one of these. New thin-film magnetic disks represent a major advance in the design and production of rigid-disk recording technology. An article by Johnson describes this new technology.

Power and Packaging

The system power control network (SPCN) provides an independent serial network that connects the operating system with the power components of the Version 2 system. It provides additional capabilities for hardware management, power control, power status, and service support. Through the SPCN, the operating system is connected to the power components of various units of the system to provide for total system power control, operating system notification of power faults, and integrated battery backup control. This provides improved power control and the ability to isolate problems in the power supplies. The article by Berglund describes how distributed power, battery backup, and a network of distributed intelligent power controllers improve system availability and serviceability.

The mechanical packaging of the Version 2 system provides for the physical, thermal, and electromagnetic protection of the logic, power, battery backup, save and restore device, and the integrated disk unit. Compliant-pin technology allows the use of a double-sided backplane in which cards are plugged from both sides. This technology increases the packaging density, reduces the need for connecting cables, and provides for a solderless assembly process. The mechanical packaging is described in three articles: Przybylski and Thorvilson's; Seyfert, Squillace, and Thornton's; and Collett's.

Manufacturing Overview

IBM* Rochester has worldwide development responsibility for the AS/400 system and advanced storage devices. Rochester manufactures the AS/400 system for the United States, Canada, and Japan marketplaces. The AS/400 products are also manufactured in Europe and Mexico and remarketed through IBM business partners in Mexico, Brazil, and Japan.

The processes used to manufacture the AS/400 system are developed with the participation of multidisciplined teams from all locations.

The AS/400 system is built to a customer order. Rochester's manufacturing plant is capable of producing any configuration of any AS/400 model, depending on the customer's order. The manufacturing process emphasizes simplicity and provides flexibility through the use of primarily manual final assembly operations. The system assembly, test, code load, and packaging processes produce high quality products and have a total elapsed cycle time (start of build to packaged machine) of less than one day. Four articles have been chosen to describe key portions of the manufacturing process (see Figure 3 on page 8) that contribute to the overall process quality and efficiency.

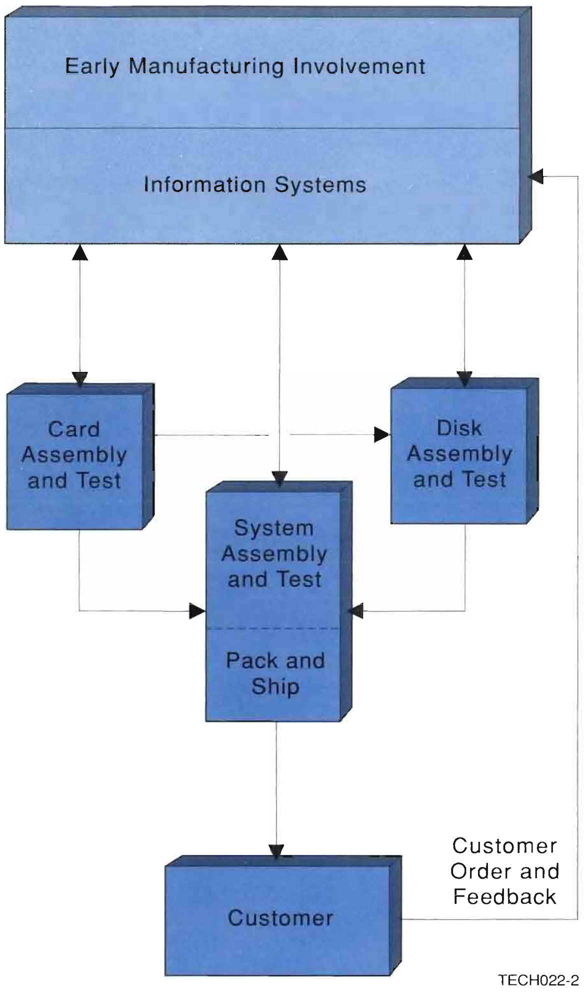


Figure 3. *Manufacturing Process Flow*

Paske, Wheeler, and Rommel describe how development and manufacturing teams interact to produce product designs that emphasize quality and manufacturability.

Guibert and Stowe detail the processes used to assemble the AS/400 system and preload software into an integrated package at the plant. Included is a discussion of the process development laboratory that describes how processes are developed and tested before being used for volume production.

Halphide and Kearns describe how early involvement and enhanced test equipment and processes enable the manufacturing group to deliver high quality products at a lower cost.

Saltness and Horejsi discuss the flow of information in the card assembly area and how the manufacturing floor control system has changed to match the flexibility required in today's manufacturing environment.

Conclusion

The AS/400 system continues to provide new technologies in hardware, software, and manufacturing while preserving customer investment in education and application programs. The proven architecture of the AS/400 system not only allows the incorporation of new technologies to meet today's customer needs, but enables the incorporation of future technologies as they become available.

The articles that follow address some of the technologies that have been incorporated since the AS/400 system was first introduced.

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.

AS/400

PROGRAMMING

Operational Assistant User Interface

Describes a new OS/400 user interface, which simplifies common system tasks yet provides access to the full power of the system.*

Julie A. Ransom, Dennis J. Schmidt, and
Timothy J. Massaro

Introduction

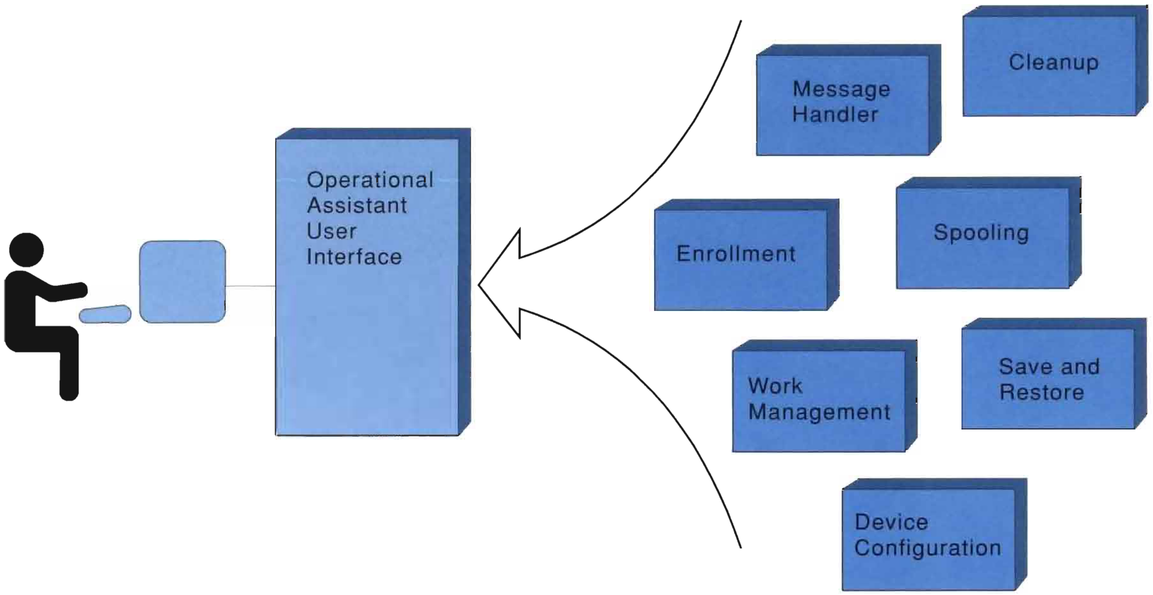
Back around the turn of the century, a new machine, the automobile, was just beginning to attract the attention of a few interested onlookers. Operating one of these vehicles was beyond the abilities and desires of most people. “Who would ever want, or even need, to drive one?” Automobiles were hard to start, hard to keep running, and, furthermore, every vehicle was different from the next one, so that learning to drive one vehicle was no guarantee of knowing how to drive a second one. Oftentimes, a mechanic would ride along with the driver to handle any of the numerous mechanical failures.

We have come a long way since those days. So far, in fact, that most people today can drive and do so without giving the process a second thought. We are able to drive as we talk, read maps, or eat a sandwich. Perhaps you can remember completing a trip by car, but you can’t specifically remember stopping at certain red lights, making all the turns, or reading individual road signs. Yet you are reasonably confident that you did everything correctly. What made it possible to go from the days of hand-cranking, mechanic-aided, manual-controlled driving to the modern era of driving as a nearly instinctual reflex? The answer is standard and simple user interfaces. Many of the difficult and complex tasks, such as starting the car or controlling the choke, have been automated or greatly simplified. Power steering, automatic transmission, and power brakes have made driving less physically demanding. And, standard controls allow people to drive nearly any car after learning to drive a single automobile.

People drive automobiles to quickly get from one place to another, so they can take part in some activity at the new location. Thus, the car is a transportation tool or aid, and driving is not an end in itself, but a means to an end. Likewise, computer users want their computers to be a tool, like a telephone or a copier, that helps them do their job. Many users typically want the benefits of using a computer to assist them, but they often do not have the technical background of the traditional computer user. This type of user should not need to give the interface a second thought. Just like the automobile driver who merely wants to run an errand across town, unaware of the workings of an internal combustion engine, computer users want to accomplish useful work, without understanding the intricate language of their computer.

The Operational Assistant user interface provides more ease of use and less complexity than the interfaces of most computers. This interface is designed to be simple by eliminating complex and technical concepts, to be automatic by providing the capability to do certain tasks for the user, and to be minimized by including only the most highly used functions (those system tasks done 80% of the time). User’s actions are anticipated and users are guided to correct solutions. Those tasks that can be automated are no longer a user’s responsibility. There is also a means to access the full capability of the system for users who become more comfortable and more familiar with the interface and want to take advantage of more advanced functions.

The resulting Operational Assistant user interface makes the end user more productive, frees up the technical support personnel, and minimizes the training needed to successfully use the AS/400 system. This user interface (see Figure 4 on page 11) provides an easy interface to system tasks, just as today’s cars simplify the complexity of internal combustion engine automobiles.



TECH104-2

Figure 4. *An Easy Interface to System Tasks*

The Operational Assistant user interface focuses on system tasks commonly done by end users and system operators, such as:

- Working with printer output
- Working with batch jobs
- Sending and receiving messages
- Saving information about a problem
- Working with device status
- Backup tasks
- Enrolling and removing users
- Cleaning up the system
- Powering on and off
- Changing system options

Let's take a look at some of the key aspects of the Operational Assistant user interface.

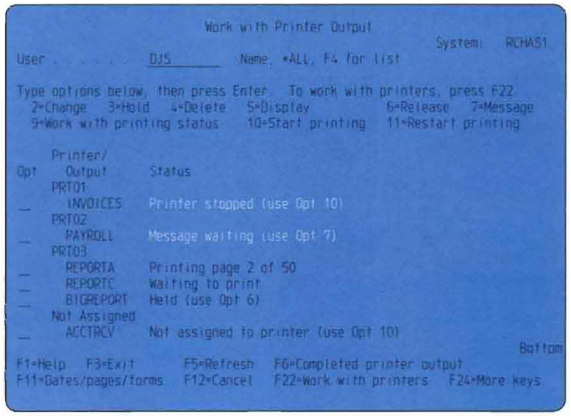
Simplifying the User Interface

One of the first objectives in the design of the Operational Assistant user interface, is making the user interface simple. Making it simple means getting back to the essentials and making it more understandable. This was accomplished by focusing on concepts and terminology.

Concepts

One way to keep things simple is to keep the concepts the user deals with simple. For example, if a user understands what a printer is, that printer output prints on a printer, and that the output must wait its turn to print, then only show those ideas. Do not expose the internals of the system like output queues, spooled files, or writers. Only show the things the user can see and touch, such as printers and printer output.

Figure 5 is an example of organizing information that corresponds to a user's concept of how things work. The printer output is organized by the printer it will print on.



```
                         Work with Printer Output
                                                          System:   RCHAS1
User . . . . . . .   DJS           Name, *ALL, F4 for list

Type options below, then press Enter.  To work with printers, press F22.
  2=Change   3=Hold   4=Delete   5=Display            6=Release   7=Message
  9=Work with printing status    10=Start printing    11=Restart printing

     Printer/
Opt    Output       Status
     PRT01
_      INVOICES     Printer stopped (use Opt 10)
     PRT02
_      PAYROLL      Message waiting (use Opt 7)
     PRT03
_      REPORTA      Printing page 2 of 50
_      REPORTC      Waiting to print
_      BIGREPORT    Held (use Opt 6)
     Not Assigned
_      ACCTRCV      Not assigned to printer (use Opt 10)
                                                                      Bottom
F1=Help   F3=Exit         F5=Refresh   F6=Completed printer output
F11=Dates/pages/forms     F12=Cancel   F22=Work with printers   F24=More keys
```

TECH129-0

Figure 5. *Work with Printer Output Display*

Terminology

The words used in a user interface can also make a big difference to users. Choosing words that users already understand makes learning to use the interface easier.

Figure 5 contains some examples of terminology changes for printer output. The AS/400 system uses the term **spooled files** to describe things that are printed; these are stored in **output queues**. These terms are understood by programmers; however, testing and comments from users indicate that many users who are unfamiliar with computers do not understand these terms. The Operational Assistant user interface uses the term printer output instead of spooled file. The term output queue is not used because it is equivalent to a user's concept of a printer. In the Operational Assistant user interface a piece of output is associated with a printer, not an output queue.

Minimizing the User Interface

Hand in hand with the concept of keeping things simple is keeping them to a minimum. The fewer things users must choose from, the more likely they are to succeed in completing their task. This is accomplished by focusing on the number of tasks, prompts, displays, options, and steps presented to users.

Fewer Tasks

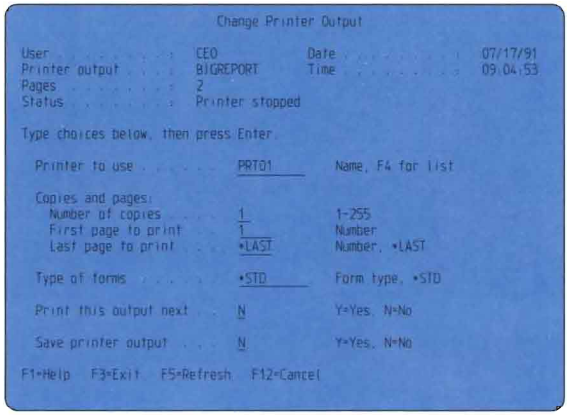
Limiting the number of tasks a user must choose is one way to effectively minimize the interface. For example, the Operational Assistant menu, accessed by pressing the Attention key (Attn) or typing GO ASSIST, provides only the system options the user typically uses, not all the functions of the system available to the user.

Fewer Prompts and Displays

Using fewer prompts reduces the amount of information users have to remember and understand. In the example in Figure 6, only the most commonly used prompts are shown (only seven). Compare this with the command to change printer output that has 28 parameters, some of which require multiple values.

By using as few prompts as possible, all those necessary may fit on one display. When the number of prompts to work with is relatively small, users do not have as many choices to make. Another benefit of limiting the number of prompts is that users are less likely to forget to specify a value.

The fields are organized so that information that cannot be changed is at the top. The prompts that can be changed are organized from the most often changed at the top to the least often changed at the bottom.



```
                         Change Printer Output

User . . . . . . . :   CEO             Date . . . . . . . :   07/17/91
Printer output . . :   BIGREPORT       Time . . . . . . . :   09:04:53
Pages  . . . . . . :   2
Status . . . . . . :   Printer stopped

Type choices below, then press Enter.

  Printer to use . . . . . .   PRT01          Name, F4 for list

  Copies and pages:
    Number of copies . . . .   1              1-255
    First page to print  . .   1              Number
    Last page to print . . .   *LAST          Number, *LAST

  Type of forms  . . . . . .   *STD           Form type, *STD

  Print this output next . .   N              Y=Yes, N=No

  Save printer output  . . .   N              Y=Yes, N=No

F1=Help   F3=Exit   F5=Refresh   F12=Cancel
```

TECH130-0

Figure 6. *Change Printer Output Display*

Fewer Options

Using fewer options limits the number of choices available to the user. This helps prevent errors caused by uncertainty about which option to choose (if options are similar) and also reduces the time needed to choose the correct option. For example, when working with printers, users have options such as start, stop, hold, release, and change. The options to change, hold, and release are not included in the Operational Assistant user interface because they are not typically needed. This makes choosing an option from the remaining options easier.

Fewer Steps

Reducing the number of steps helps the user to successfully complete the task. Each step of a task introduces the possibility of an error by the user. The greater the number of steps, the greater the chance for error. Also, a large number of steps can introduce impatience, frustration, and a lack of confidence in the user, which in turn may lead to errors of omission and oversight, or to the decision not to use the system. Following is an example of reducing the number of steps required to successfully complete a task. In the example, an Operational Assistant user chooses only one option instead of a series of steps to successfully complete the task.

To get printer output to print on a printer, a user may need to do each of the following:

1. Make sure the spooling subsystem is started
2. Make sure the spooling job queues are released
3. Make sure the output queue is released
4. Make sure the spooled files are released
5. Start a writer from the output queue to the printer or release the writer

Option 10 (Start printing) on the Work with Printer Output display (see Figure 5 on page 11) performs all of these steps.

Anticipating and Guiding Users

Anticipating a user's next move and guiding a user to the successful completion of a task is another important principle used in designing the Operational Assistant user interface. By anticipating a user's actions, the interface can more likely provide the users with the correct information and choices needed to continue to the next phase of activity. The more the interface guides users, the less they have to search for correct choices. The result is less frustrated users and a higher rate of successful tasks.

Some of the tangible ways to anticipate and guide users is to provide highlighting, recommend actions, confirm actions, and provide available choices.

Providing Highlighting

Highlighting is a means of emphasizing something important. This can be accomplished by using bold or reverse image characters. The Operational Assistant user interface uses bold characters on monochrome displays and white characters on color displays to highlight important information.

For example, Figure 5 on page 11 shows an example of highlighting statuses that require actions, such as **Message waiting (use Opt 7)**. Highlighting of statuses helps the user locate entries in a list for which they need to take an action. However, not all statuses need to be highlighted. Statuses intentionally caused by users, such as `Held (use Opt 6)` are not highlighted.

Recommending Actions

Once a user identifies that an action is required to correct a situation, the next step is to determine what needs to be done. The correct action to take may not be straightforward or obvious. Figure 5 on page 11 shows an example of a recommended action. The action is placed in parentheses and added to the end of the status, for example, `(use Opt 10)`.

Confirming an Action

Some actions result in unpleasant consequences for users, such as deleting an inventory file that was just updated. For actions that may have unwanted or unexpected results, a confirmation display is shown that allows a user to confirm that the action requested is truly the correct action.

Examples of actions that require user confirmation are deleting things and starting an activity that may take a long time, that does more than users expect, or that may be difficult to recover from, such as powering off the system.

Providing Positive Feedback

With any action a user takes, feedback is provided about what happened. Errors provide a message, window, or display that explains what happened and what to do next. Just as important as providing information about errors is providing information that the action completed successfully. Providing positive feedback leaves no doubt in a user's mind about what happened. For example, `Attempting to Start`is shown for the status after the start printing option is used on a piece of printer output.

Sometimes an action may take a long time to complete, such as building a long list of information. When these situations occur, users need feedback, while the action is running, that everything is okay and making progress. Providing periodic status messages is the best way to accomplish this. Another method is to display a window with the message that the action is in progress and to please wait. If an action may take a significant amount of time to complete, the action is confirmed before starting it.

Guiding Users through a Task

Whenever many steps are necessary to complete a task, the possibility of not successfully completing the task increases. Making a mistake in any one step can lead a user down the wrong path. Just needing to know what the correct steps are and in what order they need to be performed can be too difficult for many users.

If multiple steps are necessary for a user to successfully complete a task, the interface should guide the user through the steps.

An example of guiding a user through such a task is using option 10 to start printing in Figure 5 on page 11. Using this option causes another display to appear that shows the list of printer output that will print and the type of forms the printer will use. This guides the user through starting the printer.

Providing Available Choices

Showing the allowed choices is important for any prompt where a user must specify a value. Typically, the possible choices are to the right of the prompt. However, sometimes the list of possible choices is dynamic, such as when specifying a library, printer, or tape device. In these cases, it is helpful to provide a means to present a list of possible choices that are currently available, so the user does not have to exit the display to find needed information. The *User* prompt in Figure 5 on page 11 and the *Printer to use* prompt in Figure 6 on page 12 are examples of providing a function key (F4) to list the possible choices.

Also, if there is only one possible choice for a prompt that can have dynamic values, the Operational Assistant user interface provides that value. For example, if only one tape device is on the system, the prompt defaults to that name rather than forcing the user to enter the name.

Providing Helpful Information

When helping users do any task, the Operational Assistant user interface provides the information needed to successfully complete the task. For example, the list of possible printers also shows the status, type of form being used, and description for each printer. Because more than just the printer name is provided, the user has a better chance of choosing the correct printer.

The Operational Assistant user interface provides help information for each area of each display. But this interface is also the first portion of the OS/400 operating system to provide hypertext help. **Hypertext help** is used to provide a weblike structure of information nodes linked together by associations that allow users to freely select nodes of interest. Using hypertext, this interface can both mask off technical information in help, and allow links to related information for those who want more information but are not sure where to start looking.

Automation

Automation suggests the system do as many functions as possible without requiring any action from the user. This keeps the user from having to perform certain tasks, especially those that can require a lot of effort, that need to be done often and are repetitive, and that, when done manually, are very conducive to errors. In many cases, the users do not need to be aware of these tasks unless they have a specific need or desire to know.

Examples in the Operational Assistant user interface include the power on and off scheduler and system cleanup. Once the scheduler and cleanup are set up by the user, they automatically power on and off the system, and, after a specific number of days, they delete old job logs, messages, and so forth.

Access to More Advanced Functions

Increased knowledge and improvement in job skills are natural progressions of doing almost any task. The Operational Assistant user interface accounts for that improvement and allows a smooth transition to more advanced user interfaces for users who become experts in parts, or all, of their systems.

The Operational Assistant user interface, in general, covers those system tasks done 80% of the time by 80% of the users. Users need to have access to interfaces that provide all the functions of the system for those instances when they need to do something out of the ordinary.

Assistance Levels

The primary method for allowing access to more technical functions is the assistance level support. This allows users to select the interface they want to use in their interaction with the AS/400 system. The various interfaces are referred to as assistance levels (basic, intermediate, or advanced). At the basic level is the Operational Assistant interface. The intermediate level provides access to the full function of the system. And, the advanced level is similar to the intermediate level except that the text for options and function keys is removed to provide as much information as possible on a display.

From the user's point of view, the system provides multiple user interfaces or levels of assistance from which to choose. Each user can select a different user interface for each of several commands that support multiple user interfaces. The system retains which user interface the user wants to use for each of those commands.

For each command, users can control which interface they want in one of two ways:

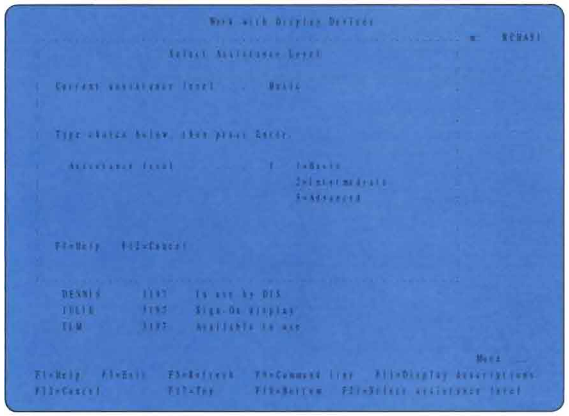
- On the command that accesses the function
- While using a function

Specifying an Assistance Level When Accessing a Function

Each command that has multiple user interfaces has an additional parameter called assistance level (ASTLVL). The assistance level parameter allows the user to specify a user interface when using the command. The default is to use *PRV (previous), the level that is retained by the system for this user and command (see "Controlling Multiple User Interfaces"). The other possible value is *USRPRF, to use what is specified in the user profile, that is, *BASIC, *INTERMED, or *ADVANCED.

Changing the Assistance Level While Using a Function

Users can change user interfaces while using a function by pressing F21 (Select assistance level). Pressing F21 causes the window in Figure 7 to appear.



```
                    Work with Display Devices
 ..................................................... :   RCHAS1
               Select Assistance Level                   :
 : Current assistance level . . . :   Basic               :
 :                                                        :
 : Type choice below, then press Enter.                   :
 :   Assistance level . . . . . . . 1   1=Basic           :
 :                                      2=Intermediate    :
 :                                      3=Advanced        :
 : F1=Help   F12=Cancel                                   :
 :........................................................:
    DENNIS       1197    In use by DIS
    JULIE        1195    Sign-On display
    TIM          1197    Available to use
                                                          More...
 F1=Help   F3=Exit   F5=Refresh   F9=Command line   F11=Display descriptions
 F12=Cancel          F17=Top      F18=Bottom   F21=Select assistance level
```

TECH131-0

Figure 7. *Assistance Level Window*

When changing the level, the system retains what the user selected. The next time this function is accessed, the user interface for the assistance level previously selected is displayed. This level is kept specifically for this function and user.

In fact, the system only retains the new assistance level when it is selected by the user in this manner. That is, when the assistance level parameter is used, the system does not retain the new assistance level. This avoids the problem of an application coding a value and changing the value retained by the system without the user consciously making the change. Another feature is that the application provider does not need to decide which interface a user should see. An application simply runs the system command and uses the default for the assistance level parameter.

Controlling Multiple User Interfaces

The system retains and controls what user interface users see with several pieces of information:

- A system value specifies the user interface for the entire system.
- The user profile specifies the user interface for a specific user.
- The interactive profile specifies the user interface for a specific command for a specific user.

Refer to Figure 8 on page 16 for a view of how the system determines what interface the user wants.

When a user enters a command or runs a command using a menu option and *PRV (previous) is specified for the assistance level parameter, the system:

1. Checks the **interactive user profile**. This is an area of storage unique for each user profile on the system. If the user ever pressed F21 for this command, an entry is stored here indicating, for this command, what level the user should see. If there is no entry here, the system continues to step 2.

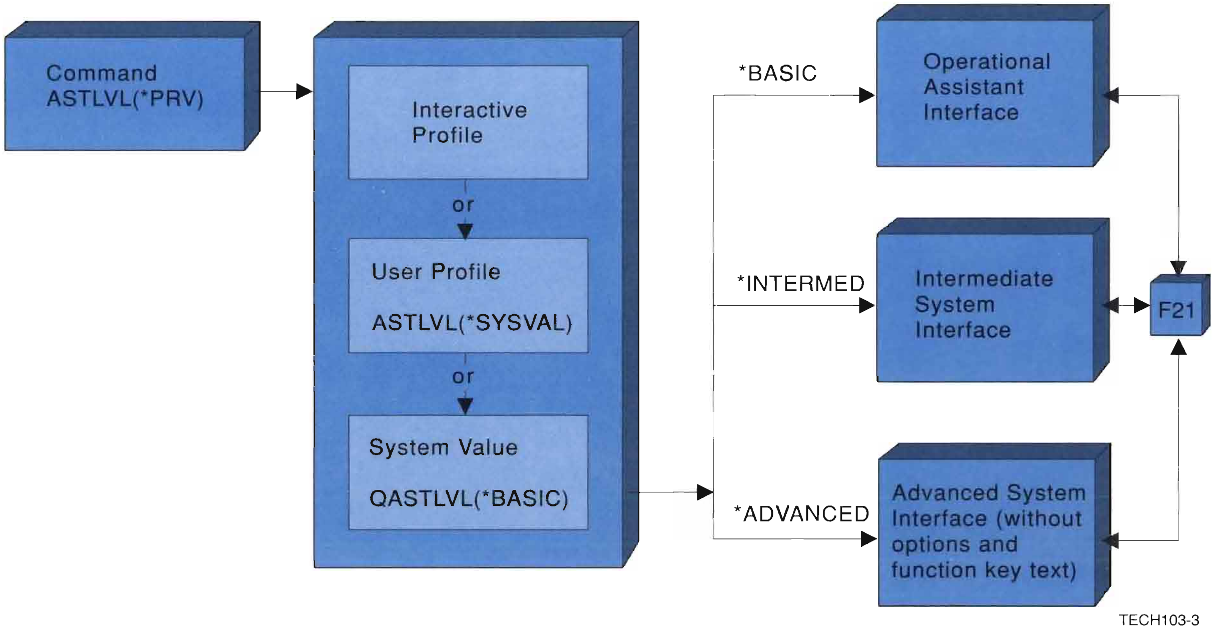


Figure 8. *Assistance Level Control.* This diagram shows an overview of how one command can access multiple user interfaces.

2. Checks the user profile itself. There is a value stored that indicates the default assistance level for this user. If this value is *SYSVAL, the system continues to step 3.

3. Checks the system value. The system value QASTLVL assistance level indicates the default for all users on the system. This value is shipped with *BASIC, meaning that the simplest interface is shown until a user requests a more complex interface.

Conclusion

User interfaces should be simple and only include the most common tasks and items users need to see or change. They should provide menus and displays that are task-oriented and should avoid using data processing terminology, while at the same time provide better and clearer messages and online help information. The interface needs to minimize the number of steps required to complete a task, the number of decisions the user must make, and the number of possibilities the user must remember. It should anticipate users' actions and guide away from mistakes, failures, and frustrations. It should lead users to satisfaction by providing positive feedback and by giving them all the information they need to successfully complete a task. The interface needs to be automated yet allow users control over the automation so they feel in control. And it should allow for the growth in the skill level of its users, and provide a way to take advantage of the full capability of the system.

Acknowledgments

The authors would like to thank the entire cross-functional team that researched, designed, and developed the Operational Assistant user interface. We would also like to thank Gary O. Larson for helping us find the words to express the ideas.

Reference

1. Larson, G.O. and Schmidt, D.J., “Designing User Interfaces: AS/400 Operational Assistant Case Study,” *IBM Technical Report TR 07.1567.* October, 1991.

Graphical User Interfaces

Describes techniques used to create graphical user interfaces for AS/400 applications.

Dana M. Duffield, Karen S. Eikenhorst, and Paula H. Richards

Introduction

Graphical user interfaces (GUIs) provide a significant increase in productivity for the end users of a system. Using GUI platforms, many applications can share the same workspace, giving end users the ability to view and work on several tasks concurrently. The consistency of GUIs gives applications an integrated appearance and decreases the learning time for new applications.

Cooperative GUI applications, which combine the strengths of the AS/400 system and a programmable work station (PWS), can be built today. Creating a cooperative GUI application enables users to leverage their current hardware and software investments and provide enhanced coexistence between the AS/400 system and a PWS.

The main elements from which a GUI is constructed are windows, icons, menus, and pointers (see Figure 9 on page 19). These elements work together to provide users with a consistent, intuitive, easy-to-use interface to their tasks.

The Common User Access* (CUA*) advanced interface guidelines define the appearance and functions of these elements in a set of user interface models. (See the *Systems Application Architecture*: Common User Access Advanced Interface Design Guide,* SC26-4582, for more information.)

Models of Graphical User Interface

From an application developer's perspective, there are three models for implementing GUIs to AS/400 systems:

- Distributed data
- Distributed display
- Distributed function

These models represent cooperative applications, which, in this context, means that the processing for the application is split between the PWS (client) and an AS/400 system (server).

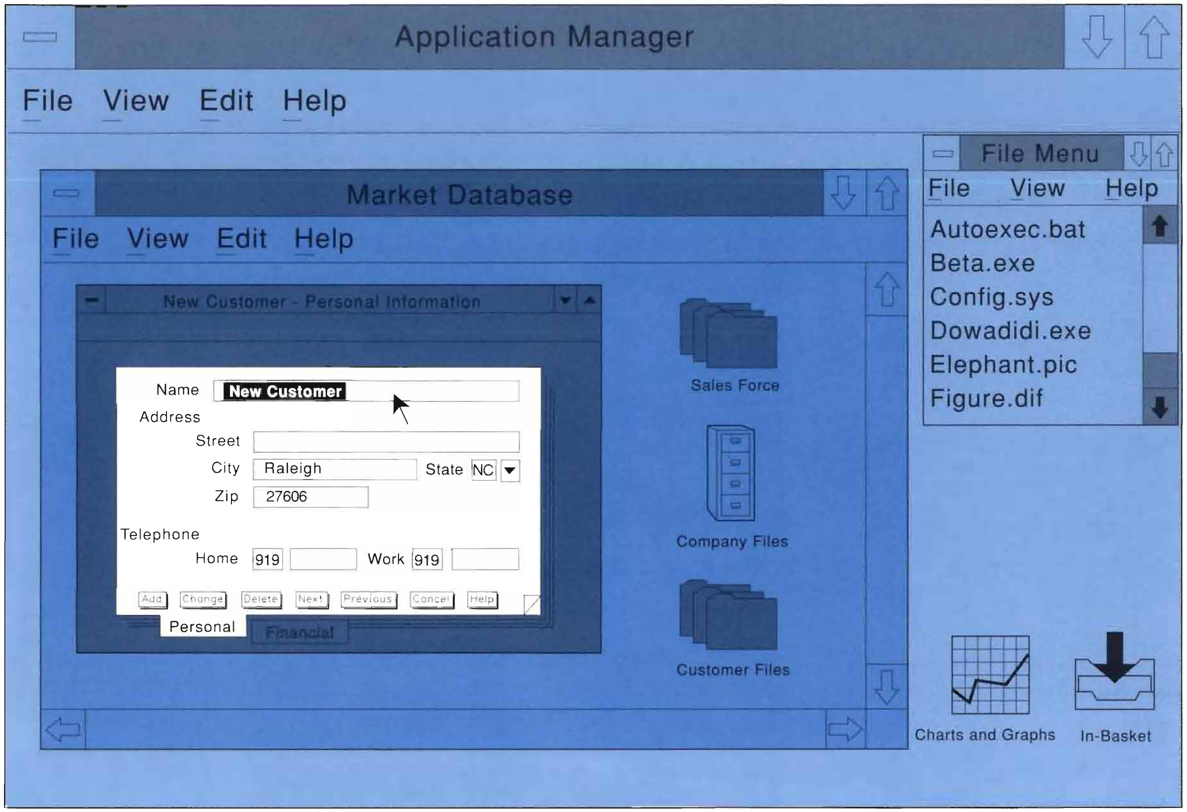
The PWS provides the capabilities for a state-of-the-art graphical user interface. The AS/400 system provides the database, backup and recovery, security, and connectivity facilities. Together, they provide a flexible, functionally rich environment for mission-critical computing.

The most common PWS platforms are Microsoft* Windows 3.0 for DOS and Presentation Manager* for Operating System/2* Extended Edition (OS/2* EE), although GUIs can be developed in DOS alone. Both the OS/2 and Windows 3.0 platforms use dynamic link libraries (DLLs) to facilitate sharing common code modules. DLLs provide support for frequently used functions, such as memory allocation, window management, communications calls, and other system APIs. (For additional information on AS/400 cooperative processing tools, see the article "Cooperative Processing" on page 52.)

Distributed Data Model

In this model, the data resides on the AS/400 system, but all of the application development is done on the PWS. The data is copied to the PWS to be displayed and manipulated in a spreadsheet, graph, or personal computer database application.

This model is a good choice for database decision support or executive information applications in which easy access to a variety of status information is required. Examples include accumulation of billed time, number of orders, and number of customer orders. This data can be retrieved from a multitude of databases, including the AS/400 system, to integrate data into a single workplace for the user.

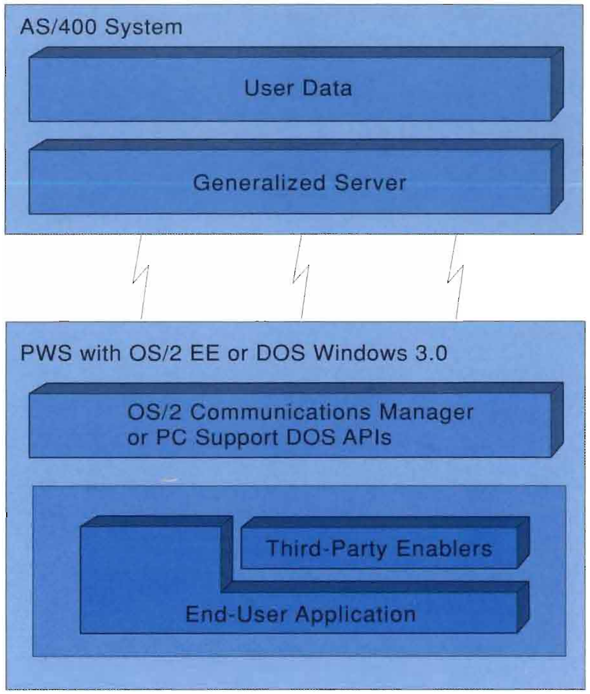


Name New Customer
Address
Street
City Raleigh State NC
Zip 27606
Telephone
Home 919 Work 919

TECH065-4

Figure 9. *Example of a Graphical User Interface to a Database Application*

This model can be implemented in a number of ways (see Figure 10). One way is to develop a PWS interface that directly issues database requests to the AS/400 system using a variety of PC Support/400 or OS/2 application programming interfaces (APIs). The application can also be created using a third-party application development tool or environment, which handles communications between the PWS and the AS/400 system and facilitates the submission of requests to the AS/400 database. Depending on the product used, the user can create macros for frequently used queries and represent them by icons, can create a structure to facilitate interactive queries, or can develop a complete graphical interface with dialog boxes, pushbuttons, and other CUA constructs.


AS/400 System
User Data
Generalized Server
PWS with OS/2 EE or DOS Windows 3.0
OS/2 Communications Manager or PC Support DOS APIs
Third-Party Enablers
End-User Application

TECH032-2

Figure 10. *Components of the Distributed Data Model*

Distributed Display Model

In this model, a GUI application is created for a specific AS/400 application that was designed for a nonprogrammable work station (NWS). The GUI application interacts with an NWS-emulated display on the PWS that is running the AS/400 application. The GUI application translates requests from the user (such as pressing a pushbutton) into keystroke instructions that have meaning to the AS/400 application. The keystroke instructions are then sent to the AS/400 application as if the user were actually typing data into an NWS-emulated display for the application. Data is retrieved from the NWS-emulated display and can be shown as part of the GUI application as text, highlighted buttons, graphics, and so on. The user is not aware of the existing application, which runs without change in an NWS emulation session.

This technique allows NWS users to continue to use the existing application, while integrating the same AS/400 application into a GUI for users of PWS devices. It is a quick way to implement a GUI to an AS/400 application because the only design work that is required is on the PWS. The existing AS/400 application code and files are used. This model can be particularly useful when adding PWS devices to an AS/400 network or when replacing NWS devices with PWS devices.

The designs of the AS/400 and the GUI applications are closely coupled because the GUI application sends specific keystroke instructions to the AS/400 application, and data is read from specific fields or locations on the display. Therefore, any changes to the AS/400 application that involve display layout, flow of displays, or keystroke interfaces require similar changes to the GUI application.

This model can be implemented in a number of ways (see Figure 11). One way is to develop a PWS interface that directly issues keystroke instructions to and reads data from a DOS or OS/2 5250 emulation product. These products include, among others, PC Support and OS/2 communications manager. Alternatively, third-party application development tools can provide the mechanism for communications between the PWS and the AS/400 system in addition to managing the synchronization of application displays. Some tools also provide the capability of converting the 5250 displays into GUI elements on the PWS. For example, they may interpret a `1.` on the AS/400 display as a menu option and automatically create a button with the name of that menu item. When the button is pressed, a 1 followed by Enter would be sent to the AS/400 application. Some application development tools also provide display design tools with object-oriented display layout utilities and may check for CUA-compliance characteristics.

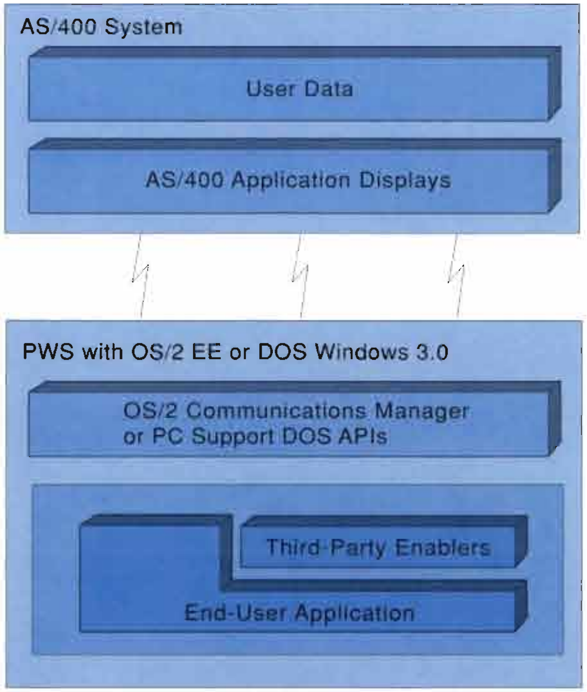

AS/400 System
User Data
AS/400 Application Displays
PWS with OS/2 EE or DOS Windows 3.0
OS/2 Communications Manager or PC Support DOS APIs
Third-Party Enablers
End-User Application
TECH031-2

Figure 11. *Components of the Distributed Display Model*

Distributed Function

When this GUI model is implemented, the function of the application is split so that each system (PWS and AS/400 system) performs the processing for which it is best suited. The PWS (client) provides the capability for an advanced user interface. The server program on the AS/400 system performs specific requests against OS/400 resources from the client program on the PWS.

The client program submits a request to start the server program, manages windows and dialog boxes, interprets buttons and mouse clicks, sends requests to the AS/400 system, and presents the results to the user. The client program can communicate with the AS/400 system using the OS/2 communications manager or PC Support APIs.

The server program is implemented in the AS/400 language of choice using an application-specific interface designed to communicate with the client program. The server program implementation is independent of the PWS platform and the design of the user interface. For example, a single server program may support both OS/2 and Windows 3.0 client programs.

Creating a GUI distributed function program requires programming on both the AS/400 system and the PWS. Using this model, the developer can design the client and server implementations for a specific set of functions.

This allows the application to be optimized for performance and provides the greatest flexibility. This model is the best choice for most GUI implementations.

There are many different client implementation options, which vary in their skill requirements, performance characteristics, and complexity (see Figure 12). These range from fairly low-level third-generation languages (3GLs), such as the C language, to higher-level object-oriented programming languages (OOPLs). The 3GLs often require the application to handle all of the window functions itself, while OOPLs may hide many of the window handling details. For many 3GLs and OOPLs, Computer Aided Software Engineering (CASE) tools are available that can assist in defining the various elements of a GUI and generating 3GL or OOPL code. In some cases these tools check for compliance with CUA standards.

Cooperative processing application generators, which may provide support for all aspects of cooperative application creation from user interface definition to host code generation and communications management, are expected to appear in the future. These products may provide a higher level of interface that will reduce the skills requirement and provide an easy-to-use platform.

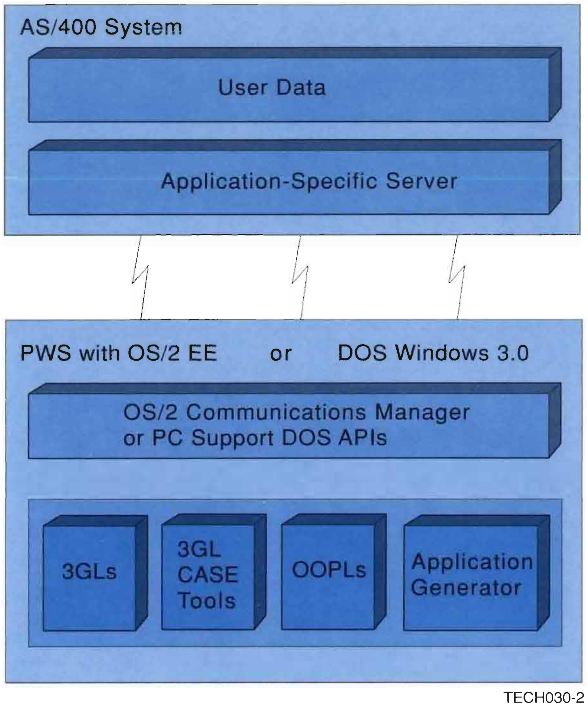

AS/400 System
User Data
Application-Specific Server
PWS with OS/2 EE or DOS Windows 3.0
OS/2 Communications Manager or PC Support DOS APIs
3GLs
3GL CASE Tools
OOPLs
Application Generator
TECH030-2

Figure 12. *Components of the Distributed Function Model*

Considerations for Developing GUI Application Programs

Developing graphical interfaces often requires a paradigm shift from standard software development. The application is usually developed on more than one platform, that is, the PWS and the AS/400 system. Many developers are familiar with one platform, but fewer are familiar with both. Depending on the implementation method chosen, both sets of skills may be required. Since all of the implementations are cooperative processing in nature, some understanding of cooperative processing techniques may be necessary.

Some understanding of object-oriented programming concepts may also be required. The design of most application software is procedural, with a set number of choices available to a user at a given time. GUIs usually provide a broader range of choices to the user at a given time and are implemented using one of the concepts of object-oriented programming known as message-based programming. With this technique, the software basically loops, responding to messages from objects such as pushbuttons and windows. When a specific message is received (such as the press of a pushbutton), the software receives the message, performs some action based on the message and data, and returns to an idle state. The user is directing the program instead of the program directing the user. This results in a software design that is different from traditional programming design.

Programming languages and tools are an important consideration in developing GUI applications. Tools can greatly increase programmer productivity, facilitate prototyping, and reduce the paradigm shift from procedural to message-based programming. However, development requirements must be fully understood to choose the correct tool. Available programming and design skills should be considered. These skills might determine the implementation model chosen or the capabilities of required tools.

PWS platforms must also be considered, and both business and technical factors affect the selection of a platform. The two most common PWS platforms are the OS/2 and Microsoft Windows 3.0. Factors that should be considered are how versatile and efficient the platform is, the availability of required functions on the platform, marketplace direction, and the hardware and software plans of a company.

The platforms for which tools are available should be considered if more than one PWS platform is targeted. If a tool or programming language is available on only one platform, but multiple PWS platforms are required, then that tool or language is not a good choice for that project. Some products require a run-time license for every PWS that the GUI application runs on. This may be an important cost consideration. Some products may also require additional resources, such as main storage or disk space to perform acceptably.

Conclusion

Cooperative GUI applications, using the combined strengths of the AS/400 system and a PWS, can be built today. Cooperative GUI applications can be implemented with several models. The model a user selects for development is based on the application requirements. Likewise, the performance and functionality of the resulting application are dependent on the model selected and the implementation decisions that are made. While it is possible to build a GUI application using only the languages available on the PWS and the AS/400 system, there are a variety of tools available to automate and simplify much of the implementation process.

User Interface Manager Help

Describes the concepts of how application developers can use the OS/400 user interface manager to provide online help and index search support for application displays.

Paul V. Allen

Introduction

Since the introduction of the AS/400 system, the Operating System/400* (OS/400*) licensed program has defined and displayed its online help information using an internal component of the operating system. The internal component is called the user interface manager (UIM). The UIM was created to easily develop consistent, easy-to-use online help for the OS/400 program. The help functions provided by the UIM are included in the OS/400 Version 2 Release 1 announcement.

Using the UIM, an application programmer can provide help information from application panels and from command prompting panels. The programmer defines the help modules independently from the application panels and commands. A **help module** is a named unit of help information that exists in a panel group object. A help module can be a single sentence or several paragraphs including lists of information.

Highlights of UIM Help

Among the key elements of the UIM are:

- Contextual help
- Help in windows
- Hypertext

Contextual Help

Contextual help provides help information for specific areas of a panel. When a user requests help using a Help key, the UIM displays the help module associated with the area where the cursor is positioned. When the cursor is in an area for which no specific help has been defined, the UIM displays extended help for the panel. When contextual help is displayed, a function key is provided to display the extended help for the panel.

Extended help typically consists of an introductory help module to describe the panel or command and how to use it, followed by each of the contextual help modules. In effect, the extended help is a concatenation of all the contextual help for a panel or command and is preceded by introductory help.

Help in Windows

The UIM displays help in a window. The UIM dynamically sizes and positions the window to show a user the underlying context from which help was requested. The intent is to position the help window as closely as possible to the contextual area for which help is being displayed without overlaying that area. In some cases, where the contextual area is large, it is necessary for the window to overlay some or all of that area.

The UIM uses an initial, or minimum, window size to determine the best location for the window. After positioning the window, the UIM may increase the size, if necessary, to take advantage of the remaining space on the screen. The UIM provides a function key, if needed, to enlarge the window so a user can see more of the help information at one time. This is useful when there is an extensive definition of a particular field, list column, or menu option.

Although the default presentation of help information is in a window, this can be overridden by an application programmer or by a user profile option.

Hypertext

The traditional approach to providing information to users is through linear definition where a reader is expected to start at the beginning and read to the end. In recent years, the concept of hypertext has gained much popularity for providing online information.

Although hypertext has only recently become popular, it is not a new idea. Vannevar Bush is usually credited with originally describing the concept of what is now known as hypertext in the article “As We May Think,” first published in the July 1945 issue of *The Atlantic Monthly*. Ted Nelson is credited with coining the term hypertext in the 1960s.

The term hypertext can be defined many ways. For the purposes of this article, **hypertext** is defined to be nonlinear information. This means that the information being provided is organized as a network of information nodes. A user reading the information controls the path through the information. This allows a user to choose the information that is needed at the time.

The UIM provides a programmer with the ability to define static hypertext links from a word or phrase in one help module to another help module. When a user selects a hypertext phrase, the UIM displays the help module identified by the hypertext link. The UIM keeps a list of the hypertext links that have been taken. When F12 (Cancel) is selected from one hypertext module, the UIM displays the most previous module from which the last link was taken.

Two navigational aids are available to a user who is “traveling” through the hypertext network. The first aid is a visual marker indicating whether or not the user has already seen the target help module. Because the information modules can be linked together in a large network, the marker helps keep the user from reading in circles.

For the second aid, F6 (Viewed topics) is available to display the hypertext path that has been taken. When a user selects this function key, a window is displayed containing a list of the help modules that have been displayed in the network. The list is displayed with the starting point help module at the top and the most recent help module at the bottom. The user may position the cursor to one of the help modules in the list and press Enter to return directly back to a specific help module.

UIM Tag Language

The help information displayed by the UIM is defined using a markup language. The markup language consists of a set of tags, attribute labels and values, and text. A **tag** is a name for a type of online information element. For example, **:help.** is the tag used to identify the beginning of a help module and **:p.** is the tag used to identify the beginning of a paragraph of text.

An application programmer provides markup for help modules in a source file member. The programmer specifies the source file member as input to the Create Panel Group (CRTPNLGRP) command. As a result of this process, a panel group (*PNLGRP) object is created in a library. The source file member is only used for creating the panel group object and is not needed at the time when the help is displayed.

An example of the UIM tag language follows with a description of the markup language elements used. This example is a portion of the extended help for the AS/400 Operational Assistant menu (ASSIST menu) in Version 2 Release 1 of the OS/400 program.

```
:HELP name='assist/help'.
Operational Assistant - Help

:P.The AS/400 Operational Assistant (ASSIST)
menu simplifies some of the common user
tasks, such as working with printer output,
jobs, messages, and changing your password.

:P.In addition, users with proper authority are
given options to help manage the system, such
as cleanup, power on and off, and system
enrollment.

:P.From this menu, you can select
the task you want to do.
.* The rest of the source is not shown

:EHELP.
```

In the above example:

:HELP. is the name of the tag used to identify the beginning of a help module. The ending period for all tags actually appears after the last tag attribute specified.

name= is the label for the attribute to identify the name of the help module being defined.

‘**assist/help**’ is the name (enclosed in apostrophes) of the help module being defined.

Operational Assistant - Help is the text for the :HELP tag. This is the title text for the help module.

:P. is the tag used to identify a paragraph of text followed by its ending period.

The AS/400 Operational... is the beginning of the text of the paragraph.

.* in columns 1 and 2 identifies a comment line in the source file member. The comment lines are ignored when the Create Panel Group command is running.

:EHELP. is the tag that identifies the end of the definition of a help module.

This example shows only three of the tags available in the UIM tag language. There are many more tags that provide other help information elements, such as unordered lists, ordered lists, examples, definition lists, and hypertext links.

Uses for UIM Help

An application programmer can use the UIM to provide online help information for data description specifications (DDS) displays and for non-IBM (third-party or user-defined) commands. In addition, the UIM can be used to create searchable help indexes similar to the index search function available from OS/400 help.

Help for Data Description Specifications

Two methods for providing help from DDS displays were provided by the OS/400 program before Version 2. The two methods are DDS record format help and OS/400 document help. Neither of these methods provides a help environment that includes dynamically sized windows, hypertext, and index search.

The UIM provides a third method for providing help from DDS. This is done through several DDS keywords. The programmer uses these keywords to establish the help module to be used for the beginning of the extended help as well as contextual help areas in the DDS display. An additional keyword can be used to specify the index search object that is available from the help panels.

Help for Commands

Prior to Version 2, there was no method available for defining help text for non-IBM supplied commands. This includes all commands created by customers, IBM business partners, and independent software vendors. With the UIM, online help can be provided for these commands. The command help is displayed when the Help key is pressed from the command prompting panels and when the Help key is pressed from a system panel where the command name is typed on the command line.

When an application programmer creates a command, two parameters on the Create Command (CRTCMD) command (shown in Figure 13), identify the help for the command.

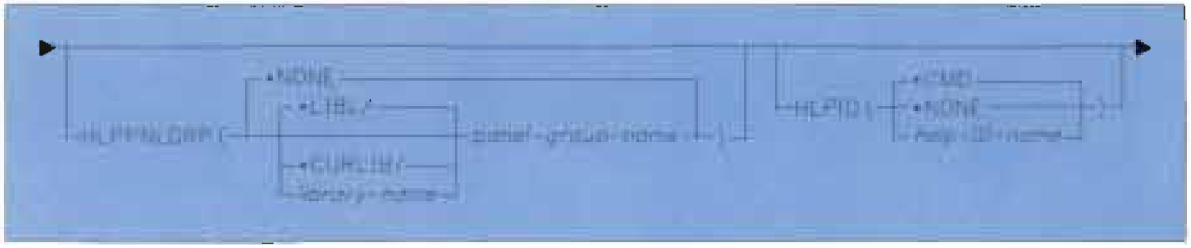


Figure 13 *CRTCMD Parameters for UIM Help*

The HLPPNLGRP parameter identifies the panel group object that contains the help modules for the command. The HLPID parameter identifies the prefix of all the help module names for the command.

Extended help for a command consists of an introductory help module followed by one contextual help module for each command parameter. There is no contextual help for command parameters that have a constant value.

The name of the introductory help module for extended help is the same as the prefix specified on the HLPID parameter of the CRTCMD command. Each contextual help module name is a concatenation of the following three values:

- The prefix specified on the HLPID parameter of the CRTCMD command
- The slash (/) character
- The value specified on the keyword (KWD) parameter of the Parameter Definition (PARM) statement in the command definition source

Index Search

An application programmer can use the UIM to build index search. The index search function allows a user to search for online help topics. The UIM performs the search based on a word string, phrase, or sentence entered by the user.

The Create Search Index (CRTSCHIDX) command creates an index search (*SCHIDX) object into which index search topics can be added. When the index search object is created, it is empty. A programmer uses the Add Search Index Entry (ADDSCHIDXE) command to add the index search topics and synonym definitions from a panel group object to an index search object.

An **index search topic** is a special type of help module in a panel group object. A topic is defined by an Index Search (**:ISCH.**) tag immediately after the :HELP. tag. The :ISCH. tag identifies one or more root words associated with the topic. A **root word** is a keyword that the UIM uses to search for index search topics.

An **index search synonym** is a word that can be entered by a user when searching for index search topics. Synonyms are defined by an application programmer using the Index Search Synonym (**:ISCHSYN.**) tag. One or more synonyms can be defined for each topic root word using the :ISCHSYN. tag.

When a user enters a search string on the Search Help Index display, the UIM searches the index for the synonyms that match the words entered. Then, the UIM searches for the root words associated with all the synonyms that were found. Finally, the UIM searches the topics that have root words that were found. A count of the number of synonym matches is kept internally for each topic found. Only the topics with the highest number of synonym matches is displayed. This narrows the list of possible topics.

Conclusion

The UIM is an alternative for providing online help for applications. While this alternative is not the correct solution for every application, it provides capabilities not previously available from the OS/400 program and is the method of choice for applications that need to provide the "look and feel" of the OS/400 program.

The key elements of the UIM can be used to provide state-of-the-art online help for application panels and commands. These key elements include:

- Contextual and extended help
- Dynamically sized and positioned windows
- Hypertext links providing access to detailed information as needed by the reader
- Searchable indexes for finding topics based on a search string or sentence

References

1. Botterill, J.H., Charland, D.A., and Harrington, J.Y., "An Integrated User Interface," *IBM Application System/400* Technology*, SA21-9540. June, 1988.
2. Bush, V., "As We May Think," *Atlantic Monthly*. July, 1945.
3. *Guide to Programming Application and Help Displays*, SC41-0011. May, 1991.

Windows Using Data Description Specifications

Describes how windows can be created on the Version 2 AS/400 system using data description specifications (DDS). Support has been added in Version 2 to make creating and using windows much easier.

James R. Ulwelling

Introduction

Since the days of the System/38, application programmers have been using windows to improve their application programs. A window is a secondary display that can be displayed over the top of currently displayed records. A window is smaller than the actual work station screen, and can be positioned anywhere within the screen. Windows can make an application more appealing and easier to use. Help information, for example, is usually easier to use when it is displayed within a window. The user can then see both the help information and the field for which they requested help.

Although many application programs now contain windows, creating these windows has not been easy. Until now there have been two ways to add windows to AS/400 application programs: using data description specifications (DDS) for user-created windows or using user-defined data streams (UDDS). UDDS involves writing and interpreting the 5250 data stream that is sent to the work station controller. UDDS is beyond the scope of this article and, therefore, will not be discussed. Suffice it to say that UDDS is a powerful tool, but with that power comes great complexity. An application that uses UDDS controls all input and output to the screen, which means any DDS functions used must be reinvented by the application.

Note: Other products are available to create windows within applications. However, these products use either DDS or UDDS to create windows.

DDS Windows before Version 2

Creating windows with DDS before Version 2 was not an easy task. It was labor intensive and allowed only a limited amount of window capability. The steps involved in creating such windows were:

1. Code the window's border characters, locations, color, display attributes, and the window's data or fields in the DDS source. The entire area occupied by the window must either be filled with blanks or have data written to it so that all overlaid data disappears.

   The source for windows is usually separate from the rest of the application's DDS source so the windows can be stored in a separate file. This is done for several reasons:

   - With the use of the ASSUME keyword, a window can be placed over the top of currently displayed data without losing the rest of the lines that the window overlays.
   - When the window is removed, the system automatically restores the overlaid data to the display. If the DDS for the window were to be stored in the same file as the DDS for the overlaid data, the application would have to rewrite the underlying data to remove the window.
   - The same window can be used within more than one application, or in more than one part of a single application.

2. Write the window borders and window data to the display.
3. Remove a window from the display by either closing the window file or rewriting a record from the previous display file.

Limitations to Pre-Version 2 DDS Windows

Limitations to these windows include:

- The inability to dynamically position a window horizontally on the screen. An application can dynamically change the vertical location of the window by using the SLNO keyword, but horizontal positioning of these windows is impossible.
- The inability to use a subfile within a window. A **subfile** is a group of records of the same record format that can be displayed at the same time at a display station. The system sends the entire group of records to the display in a single operation and receives the group from the display in another operation. The system allows the group of records to be scrolled through if they do not all fit on the screen.

  A true subfile cannot be used within the window. An application program can simulate a subfile within a window by using arrays, but the application itself must perform all of the subfile functions, such as scrolling and relative record selection.
- The loss of attribute information for partially overlaid fields. When a window overlays part of a field that has active color or display attributes, any part of that field that extends beyond the right side of the window is displayed using the default field attributes. For example, if the first character of a red, reverse image field is overlaid by a window, the nonoverlaid part of the field appears as the default, green field. All DDS attributes are lost.

A Better Way to Create Windows

In Version 2 Release 1 the AS/400 system supports four new DDS keywords that allow windows to be created more easily. The use of these keywords simplifies window creation while providing additional window capabilities. The four DDS keywords are:

- Window (WINDOW)
- Window Border (WDWBORDER)
- Remove Windows (RMVWDW)
- User Restore Display (USRRSTDSP)

With these four keywords an application program can:

- Dynamically position a window anywhere on the screen, vertically or horizontally.
- Use DDS subfiles within a window.
- Change the characters, color, or display attributes of a window's border at run time.
- Remove one or more windows from the display and the system will restore the underlying data.
- Request the system to remove all windows from the display and to display another window, using only one output operation.
- Inform the system not to save or restore the underlying data. Windows are only removed when the application writes over them. This allows an application to be written so that it performs better. If the application rewrites the underlying display, there is no need for the system to rewrite it.
- Overlay the beginning attribute of a field with a window without affecting that field's attributes. The system moves the field's attribute to a location just after the window so that the remainder of the field is displayed correctly.

By using these keywords the task of creating windows is nearly effortless. The user codes the WINDOW keyword and its parameters on a record format, and the system displays that record in a window when the application writes it to the screen.

No special programming techniques are needed. Any AS/400 language can display a window simply by displaying a record format that has a WINDOW keyword on it.

To remove a window from the screen, an application program must either read or write to a record format that is displayed before the window, or write a nonwindow record to the screen.

The system fills the window background with blanks when a window is displayed. If the USRRSTDSP keyword is not in effect, the system restores the overlaid records when the window is removed.

No special hardware is required. DDS windows work on both programmable and nonprogrammable work stations.

The WINDOW Keyword

The WINDOW keyword is a record-level keyword that instructs the OS/400 program to display a record format within a window. The WINDOW keyword must be specified with either four parameters or with one parameter. When four parameters are specified, they inform the system where the window appears on the screen and what its size will be. This is called a **window definition**. When one parameter is specified, it informs the system of the name of the record format that defines the location and size of the window that this record will be displayed within. This is called a **window reference**. Window references allow more than one record format to be displayed within one window.

The parameters that describe the window location can be either constants or field names. Using field names allows an application to specify where the window appears at run time. This allows a window to be dynamically positioned vertically as well as horizontally on the display.

Fields in a window record are located relative to the first usable window location in the upper-left corner of the window. When a display file containing a window record is created, the system verifies that all fields within each window record fit within the specified window dimensions. A window can be thought of as a small display, and its size is determined by the parameters on the WINDOW keyword.

When the USRRSTDSP keyword is not in effect, the system saves the screen image before displaying each window. This allows the system to restore the display when a window is removed. Unless USRRSTDSP is used, the system allows a maximum of 12 windows to be displayed at any one time. DDS can define more than 12 windows, but only 12 can be displayed at the same time. When USRRSTDSP is in effect, there is no limit to the number of windows that can be displayed.

When the USRRSTDSP keyword is not in effect, windows can be removed by either reading or writing to a previous record, or by writing a nonwindow record to the screen. The system automatically restores the overlaid information.

The last line in a window is reserved as the message line, and cannot contain any fields. For example, if a WINDOW keyword is coded that specifies 10 window lines for the window, only nine of those lines can contain fields; the 10th line is the message line.

To use a subfile within a window, the WINDOW keyword must be coded on the subfile control record format. A subfile control record format is one of two record formats required to define a subfile in DDS.

The WDWBORDER Keyword

The WDWBORDER keyword is a file- or record-level keyword that allows the window border color, display attributes, and border characters to be customized. If no WDWBORDER keywords are active when a window is displayed, the window border is made up of blue dots. All colors are ignored when a window is displayed on a noncolor display station.

The WDWBORDER keyword can be specified with the color parameter, the display attributes parameter, the border characters parameter, or any combination of the three.

Option indicators can be used on the WDWBORDER keyword to allow a window border to be customized at run time. More than one WDWBORDER keyword can be active at the same time. When more than one WDWBORDER keyword is active on the same level (file or record level), the system combines the individual parameters on the keywords. When the same parameter is used on more than one WDWBORDER keyword at the same level, the parameter from the WDWBORDER keyword specified first in the DDS is used.

The WDWBORDER keyword can also be active at the file and record level at the same time. When this occurs, the WDWBORDER parameters are combined, with parameters at the record level having precedence.

The RMVWDW Keyword

The RMVWDW keyword is a record-level keyword that is used to remove all existing windows from the display before an individual window record is displayed. Only one output operation is used to perform the RMVWDW function. There are no parameters for this keyword. Option indicators can be used to control when the system should perform this function.

The USRRSTDSP Keyword

The USRRSTDSP keyword is a record-level keyword that is used on a window record to inform the system not to save and restore the underlying display for this and any following windows. When USRRSTDSP is in effect, the application is responsible for restoring the underlying display when a window record is removed from the display. This keyword allows an application to perform better by choosing when the system should and should not save and restore the contents of the display. It also allows an application to make it appear as though two different windows are active at the same time because both windows stay on the screen as control moves from one window to the other.

Option indicators can be used on the USRRSTDSP keyword to control when the system should and should not save the display contents.

Using Windows to Display Help

The WINDOW keyword can be specified on a record format that is used as a help record. The HLPRCD keyword is used to identify the record format that is displayed when the Help key is pressed. If that record format is a window definition record, the help information is displayed in a window.

If the location of the help window is specified using field names as parameters on the WINDOW keyword, the system positions the help window automatically based on the cursor location. If the Help key is pressed while the HLPRCD keyword is in effect, the application program does not receive control. Therefore, the fields used to locate the window cannot be set by the application. The system positions the help window using the following sequence of rules:

1. If the window fits on the screen below the cursor location, it is placed there. The top window border is positioned one line below the cursor location. The left window border is positioned in the same column as the cursor location if it fits. If it does not fit, the window is moved to the left until it fits on the screen.
2. If the window fits on the screen above the cursor location, it is placed there. The bottom window border is positioned one line above the cursor location. The left window border is positioned in the same column as the cursor location if it fits. If it does not fit, the window is moved to the left until it fits on the screen.

3. If the window fits on the screen on the right side of the cursor location, it is placed there. The right window border is positioned in the next-to-last column of the screen. The top window border is positioned on the same line as the cursor location if it fits. If it does not fit, the window is moved up until it fits on the screen.
4. If the window fits on the screen on the left side of the cursor location, it is placed there. The right window border is positioned two columns to the left of the cursor location. The top window border is positioned on the same line as the cursor location if it fits. If it does not fit, the window is moved up until it fits on the screen.
5. If the window cannot be positioned in any of the previous ways, it is placed in the lower-right corner of the screen.

If the WINDOW keyword on the help record specifies a field name on only the line or only the position parameter, the help window is displayed using these rules. However, the parameter specified as a constant on the WINDOW keyword is not changed.

If the WINDOW keyword on the help record does not specify field names for the line and position parameters, the help window is displayed using the line and position values specified on the WINDOW keyword.

Examples of DDS Windows

The following are examples of how to use DDS windows.

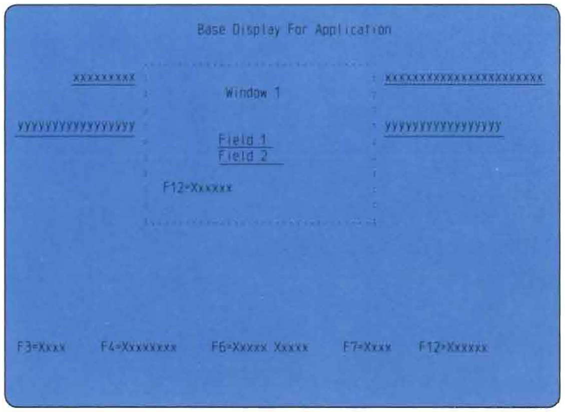
Simple Windows

The following DDS uses the WINDOW keyword to define two simple DDS windows. The window locations are constant, and the border character and color defaults are used.

```
A*
A          R BASE
A                                      CF03(03)
A                                      CF04(04)
A                                      CF06(06)
A                                      CF07(07)
A                                      CA12(12)
A                                  2 28'Base Display For Application'
A            DATA1         70A  B  5 10
A            DATA2         70A  B  8  2
A                                 22  2'F3=Xxxx      F4=Xxxxxxxx      +
A                                       F6=Xxxxx Xxxxx      F7=Xxxx    +
A                                       F12=Xxxxxx'
A*
A          R WINDOW1                   WINDOW(4 20 9 30)
A                                      CA12(12)
A                                  2 11'Window 1'
A            FIELD1         8A  B  5 10
A            FIELD2        10A  B  6 10
A                                  8  2'F12=Xxxxxx'
A*
A          R WINDOW2                   WINDOW(10 35 9 30)
A                                      CA12(12)
A                                  2 11'Window 2'
A            FIELDA         8A  B  5 10
A            FIELDB        10A  B  6 10
A                                  8  2'F12=Xxxxxx'
A*
```

Figure 14. *DDS Source Code for a Simple Window*

Using this DDS, record format BASE is written to the screen followed by writing record WINDOW1. Record WINDOW1 is displayed within a window as follows.



```
                    Base Display For Application

          xxxxxxxxx :                         : xxxxxxxxxxxxxxxxxxxxxxx
                    :        Window 1         :
 yyyyyyyyyyyyyyyyyy :                         : yyyyyyyyyyyyyyyyyy
                    :       Field 1           :
                    :       Field 2           :
                    :                         :
                    :  F12=Xxxxxx             :
                    :                         :

 F3=Xxxx    F4=Xxxxxxxx    F6=Xxxxx Xxxxx    F7=Xxxx    F12=Xxxxxx
```

TECH132-0

Figure 15. *Example of a Simple Window*

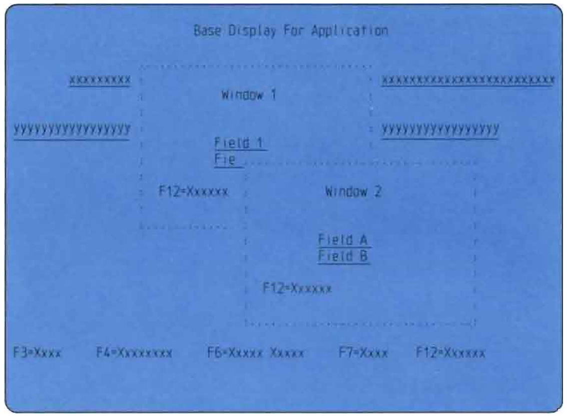
In this example, when the WINDOW1 record is displayed, the upper-left corner of the window border is on line 4 position 20 of the screen. The lower-right corner of the border is located 10 lines lower than the upper border and 33 positions to the right of the left border. The lower-right corner is calculated as follows:

- Lower border line = upper border line + window lines + 1
- Right border position = left border position + window positions + 3

Fields in a window record are located relative to the first usable window location in the upper-left corner of the window. In the above DDS source, FIELD1 has a location of row 5 column 10. When displayed, FIELD1 starts 5 lines lower than the upper border and 11 positions (the ending attribute byte for the border character has been taken into account) to the right of the left border character.

FIELD2 starts 6 lines lower than the upper border and 11 positions to the right of the left border.

When record WINDOW2 is written to the screen, it overlays the first window, and the display now looks like this.



```
                 Base Display For Application

        xxxxxxxxx                          xxxxxxxxxxxxxxxxxxxxxxxxx
                          Window 1

yyyyyyyyyyyyyyyyyy                         yyyyyyyyyyyyyyyyyy
                          Field 1
                          Fie
             F12=Xxxxxx              Window 2


                                     Field A
                                     Field B

                           F12=Xxxxxx


F3=Xxxx     F4=Xxxxxxxx     F6=Xxxxx Xxxxx     F7=Xxxx    F12=Xxxxxx
```

TECH133-0

Figure 16. *Example of Simple DDS Windows*

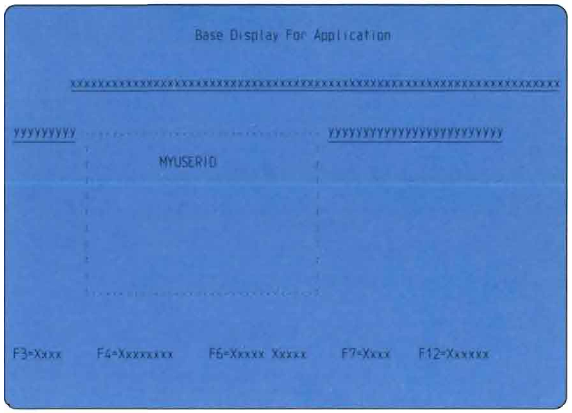
Multiple Records within One Window

The following DDS uses the WINDOW keyword to define a DDS window. The WINDOW keyword is also used to associate one more record format with the window definition. This allows both record formats to be displayed within one window at the same time. The position of the window is determined by the values of fields LINE and POS when the WINDOW1 record is written to the screen.

```
A*
A          R BASE
A                                      CF03(03)
A                                      CF04(04)
A                                      CF06(06)
A                                      CF07(07)
A                                      CA12(12)
A                                  2 28'Base Display For Application'
A            DATA1         70A  B  5 10
A            DATA2         70A  B  8  2
A                                 22  2'F3=Xxxx     F4=Xxxxxxxx      +
A                                       F6=Xxxxx Xxxxx     F7=Xxxx    +
A                                       F12=Xxxxxx'
A*
A          R WINDOW1                   WINDOW(&LINE &POS 9 30)
A            USERID         8A  O  2 10
A            LINE           2S  P
A            POS            3S  P
A*
A          R RECORD1                   WINDOW(WINDOW1)
A                                      CA12(12)
A                                      OVERLAY
A            FIELD1         5A  B  5  2
A            FIELD2        20A  B  6  5
A                                  8  2'F12=Xxxxxx'
A*
```

Figure 17. *DDS Source for Multiple Records within One Window*

Using this DDS, record format BASE is written to the screen followed by writing record WINDOW1. When record WINDOW1 is written, the application program must set fields LINE and POS to the line and position on the screen where the upper-left corner of the window should be located. For this example, assume that these values are 8 and 12, respectively. This is what the screen would look like.

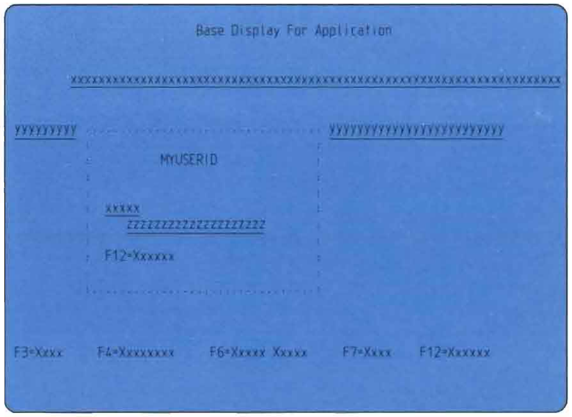


```
                 Base Display For Application

        xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

yyyyyyyyy                                yyyyyyyyyyyyyyyyyyyyyyyyy
                 MYUSERID










F3=Xxxx     F4=Xxxxxxxx     F6=Xxxxx Xxxxx     F7=Xxxx    F12=Xxxxxx
```

TECH134-0

Figure 18. *Example of First Record within a Window*

Next, record RECORD1 is written to the display. Because the OVERLAY keyword is used, both records WINDOW1 and RECORD1 appear within the window defined on WINDOW1. The screen now looks like this.



```
                 Base Display For Application

        xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

yyyyyyyyy                                yyyyyyyyyyyyyyyyyyyyyyyyy
                 MYUSERID


       xxxxx
          zzzzzzzzzzzzzzzzzzzz

       F12=Xxxxxx



F3=Xxxx     F4=Xxxxxxxx     F6=Xxxxx Xxxxx     F7=Xxxx    F12=Xxxxxx
```

TECH135-0

Figure 19. *Example of More Than One Record in a Window*

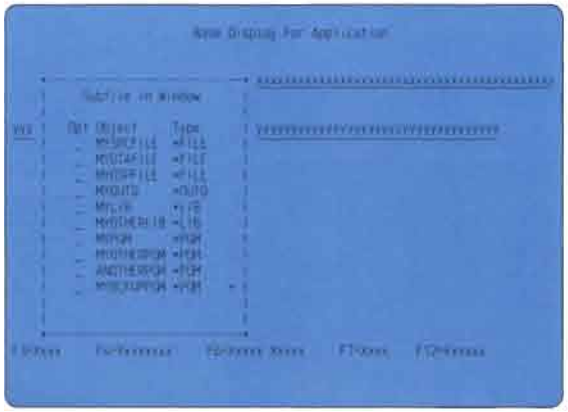
A Subfile within a Window

The following DDS uses the WINDOW keyword on a subfile control record, which causes the subfile to be displayed within a window. The WDWBORDER keyword is also used to change the border characters of the window. In this example, the WDWBORDER keyword is specified at the file level.

```
A*
A                                      WDWBORDER((*CHAR '*-*||*-*'))
A*
A          R BASE
A                                      CF03(03)
A                                      CF04(04)
A                                      CF06(06)
A                                      CF07(07)
A                                      CA12(12)
A                                  2 28'Base Display For Application'
A            DATA1         70A  B  5 10
A            DATA2         70A  B  8  2
A                                 22  2'F3=Xxxx     F4=Xxxxxxxx      +
A                                       F6=Xxxxx Xxxxx     F7=Xxxx    +
A                                       F12=Xxxxxx'
A*
A          R SFLRCD                    SFL
A            SFLFLD1        2A  I  4  4
A            SFLFLD2       10A  O  4  7
A            SFLFLD3        7A  O  4 18
A*
A          R CTLRCD                    SFLCTL(SFLRCD)
A                                      WINDOW(5 6 15 26)
A                                      SFLSIZ(30)
A                                      SFLPAG(10)
A                                      SFLDSP
A                                      SFLEND
A                                      SFLDSPCTL
A                                  1  5'Subfile in Window'
A                                  3  3'Opt'
A                                  3  7'Object'
A                                  3 18'Type'
A*
```

Figure 20. *DDS Source for Using a Subfile in a Window*

Using this DDS, record format BASE is written to the screen. The application then writes data to the subfile record, SFLRCD. Now, when the subfile control record, CTLRCD, is written to the screen, the screen looks like this.



Base Display For Application

Subfile in Window

Opt Object Type

TECH136-0

Figure 21. *Example of a Subfile in a Window*

Conclusion

Windows have become a prominent part of application programs, but until now they have not been easy to code or maintain. With the addition of the four window-related keywords—WINDOW, WDWBORDER, RMVWDW, and USRRSTDSP—windows are extremely easy to code and maintain. The time now needed to code a window using DDS is a small fraction of what it used to be. Windows can be incorporated into existing applications by selectively coding the WINDOW keyword on some record formats.

Using a subfile within a window and dynamic positioning of a window horizontally have not been supported until now. Subfiles are fully supported within these windows by coding the WINDOW keyword on the subfile control record. Dynamic positioning of windows, even horizontally, is easy to do; using DDS fields as parameters on the WINDOW keyword allows the application to determine where the window should appear.

Aesthetics are also improved because the system maintains the color and display attribute settings of all partially overlaid fields on the display. Color and display attributes no longer disappear when a window covers part of a field.

CallPath/400

Describes the CallPath/400 licensed program, which provides an application programming interface to integrate functions and information from specific telephone switches into AS/400 applications.

Laura J. Bruner

Introduction

The CallPath/400 licensed program provides an application programming interface (API) that allows applications to integrate functions and information from a variety of telephone systems. Through the API, an AS/400 application can monitor and influence the actions of an attached telephone switch. The telephone switch can be a private branch exchange (PBX) residing on a customer's premises, or other specialized telecommunications equipment. The CallPath/400 program is based on the IBM CallPath Services Architecture, which means that the API implementation is functionally consistent with implementations to be developed for other IBM computer systems. The CallPath Services Architecture provides a framework for integrating voice technology into new and existing data processing applications. The result is a highly functional set of host-based telephony services. **Telephony** refers to the transmission of voice or data communications between separate points. The AS/400 system processes messages and requests as shown in Figure 22.

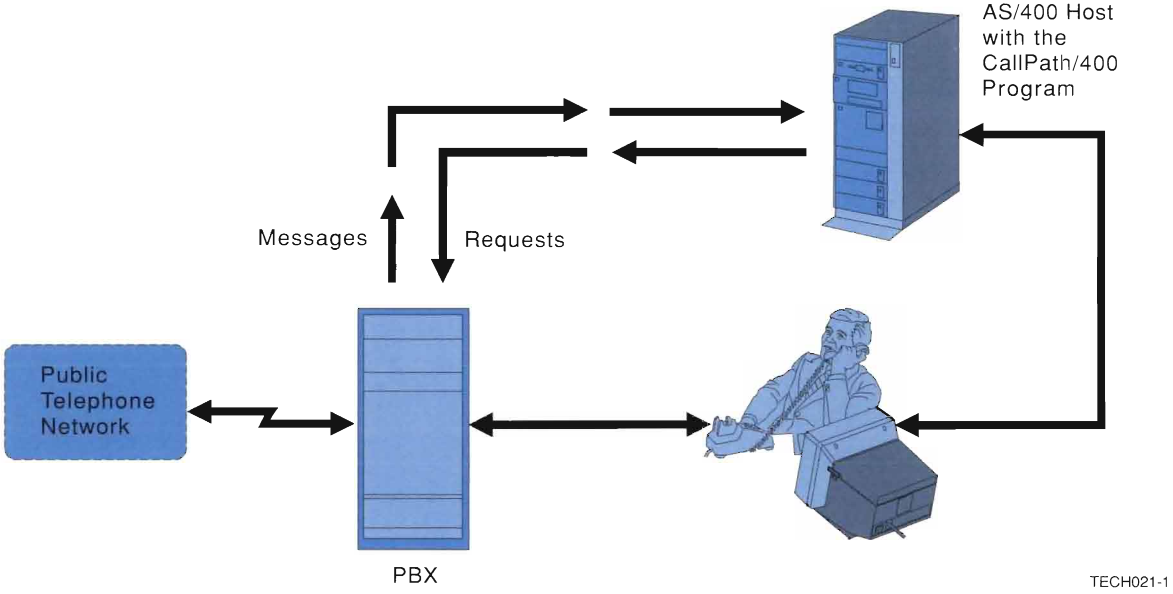


Figure 22. *Call Processing*

For example, with the CallPath/400 program, a computer application can do the following:

- Dial outbound calls
- Perform coordinated voice and data transfer
- Show customer information on the display with incoming calls
- Redirect incoming telephone calls
- Control telephone functions through the display

Application Capabilities

The following application examples illustrate the capabilities of the CallPath/400 program.

Intelligent Answering. The calling number or the dialed number information passed from the telephone switch to the CallPath/400 program can be provided in one of the available switch messages. **Switch messages** describe the progress of a telephone call as it is handled by the telephone switch. In addition, when provided by the network and supported by the telephone switch attached to the AS/400 system, the messages may contain various types of information from public telephone networks. By passing this information from the switch to the AS/400 system, the application can show an appropriate display of information. For example, a mail order company might assign a different 800 number to each of its product lines. When a customer calls, the application would use the **dialed number identification service (DNIS)** information from the switch to show the appropriate order display for that particular item. The public network provides the DNIS, which identifies a logically called party. For example, two 800 numbers might both be translated to a single real number. Using DNIS, the application can identify which of the two 800 numbers was dialed.

An application such as a customer support center might use **calling line identification (CLID)** or **automatic number identification (ANI)** information from the switch to identify customers calling for support. CLID and ANI are also numbers supplied by the public telephone network, and they identify the calling party. By receiving the switch message containing this data, the AS/400 application can attempt to match the calling telephone number to a corresponding customer database record. If a match is made, a display of information for that particular customer can be shown while the support representative answers the telephone.

Coordinated Voice and Data Transfer. An application can transfer information relating to a customer at the same time the user transfers the telephone call. For example, a customer calling the bank to order new checks may also have a question about home loans. By building this capability into the application, the telephone call can be transferred along with the customer information to a loan officer.

Personal Services. Using the API, the application program makes requests, such as transferring of calls, establishing a conference call, placing a party on hold, or retrieving a held call. Users generally initiate these requests by pressing a function key or selecting an option on their display. The application can then use switch messages to get information regarding the status of these requests.

Intelligent Dialing. If the status of outbound calls is monitored and no one answers, the application can request, using the API, that the switch disconnect the call. In the event that no one answers, the application developer can design the application to automatically initiate another call or to allow the agent to provide input before attempting another call. In addition, the user application can reschedule the calls that were not completed to be retried at a later time.

Application Programming Interface

As already discussed, the CallPath/400 program provides an application program interface (API) that allows application programs on the AS/400 system to access the services of an attached switch. The application program can make requests through the API to control telephone functions, and it can also use the API to receive switch message information. This information may be related to the status of a previous request that the application made for switch services, or it may be information about an incoming telephone call. While a large set of functions are available through the API, each switch may only offer support for a subset of these functions.

Some of the requests that an application can issue using the API include:

- **Add_Party** adds a new party to an existing telephone call.
- **Answer_Call** answers an incoming telephone call.
- **Conference_Call** joins two or more telephone calls in a conference.
- **Disconnect** disconnects a party from an existing call.
- **Hold_Call** places a party on hold.
- **Make_Call** establishes a two-party telephone call.
- **Redirect_Call** redirects an incoming call from one party to another party.
- **Retrieve_Call** reconnects a held call.
- **Transfer_Call** transfers a call from one party to another party.

Some of the switch messages that an application can receive include:

- **Call_Alerting** indicates that a telephone call is assigned to a telephone and the telephone is being alerted (ringing).
- **Call_Conferenced** indicates that two telephone calls are joined in a conference.
- **Call_Connected** indicates that a party has become an active participant in a telephone call.
- **Call_Held** indicates that a call was placed on hold by one of the parties in the call.

- **Call_Rejected** indicates that a telephone call was not completed.
- **Call_Routed** indicates that an incoming call was routed to the called party.
- **Call_Transferred** indicates that a telephone call was transferred from one party to another party.
- **Disconnected** indicates that a party is disconnected from a telephone call.

The function of requesting services of an attached telephone switch using the API is also referred to as **call control**. An example of a call control request is to establish a two-party telephone call. The user may make this request by pressing a predefined function key (or command key) or by entering an option in a field. In some applications, the application automatically starts the request based on where the user is within the application rather than requiring the user to enter input or press a key.

The application makes the call control request by simply issuing a program call to the required API program for that function. The user application must pass the parameters for the particular program call as defined by CallPath Services Architecture. The API performs validation on the parameters and indicates the results to the user program in the form of a return code parameter. If no problems are encountered, the request passes to the telephony subsystem where additional processing of the request takes place.

The API permits access to advanced network services, such as those offered by an integrated services digital network (ISDN), when they are supported by the telephone switch. The application program receives available switch messages through one of the API program calls. Messages can be generated as a result of a previous request, or they may contain information about an incoming telephone call.

CallPath Services Subsystem

The CallPath Services subsystem supports requests issued by the user application and messages received from the switch. The CallPath/400 program implements the CallPath Services subsystem and processes call control requests received from the user application and messages received from the switch by means of the **connection manager**. The connection manager is started using the Start Telephony Connection Manager (STRTELCNNM) command (provided with the CallPath/400 program) and establishes communications between the connection manager and the attached telephone switch. The type of communications support required depends on the type of telephone switch the connection manager is communicating with.

In effect, the connection manager shields the user application program from the specific communications support required and the switch-specific protocol mapping that is required. The connection manager translates API requests issued by the user application into a format that the particular target switch can interpret. In addition, the connection manager converts any messages the switch sends into a common format used by the API that the application can interpret. This structure is shown in Figure 23 on page 39.

Standards Activity

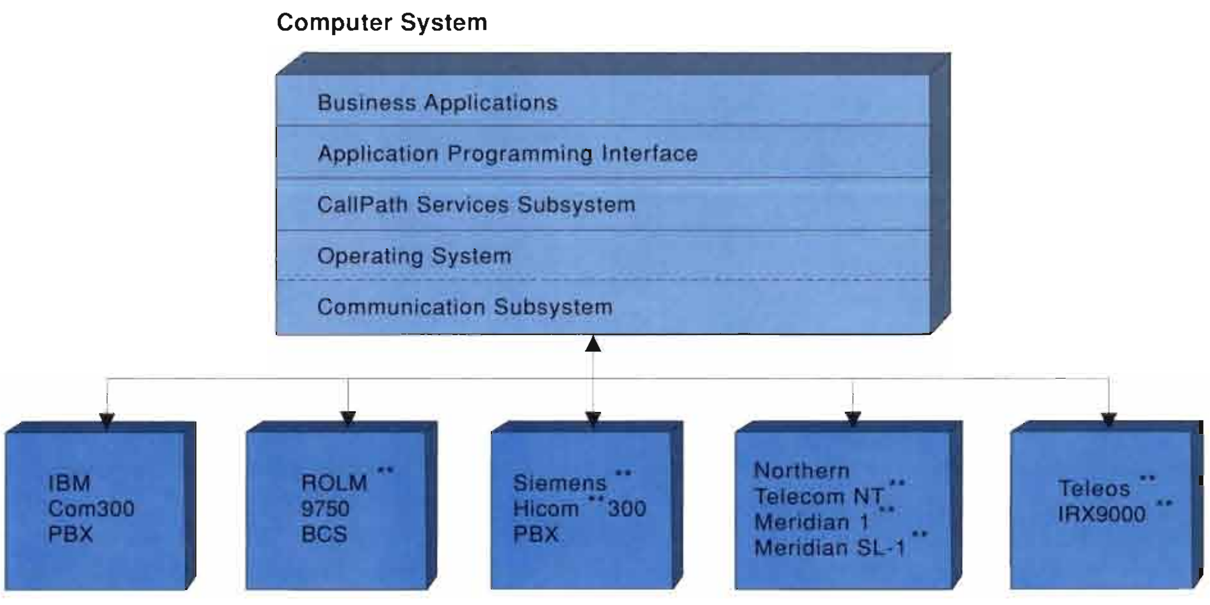
The telecommunications industry is moving toward a common, standardized protocol that is supported by many computer and switch manufacturers. IBM supports this effort and is actively participating in the following national and international organizations:

- American National Standards Institute (ANSI)
- European Computer Manufacturer's Association (ECMA)

When standards are approved and implemented, IBM intends to provide mapping between the CallPath Services Architecture API and the standard-defined formats and protocols. Mapping to the evolving standards will be provided in the CallPath Services subsystem, and it is expected that this will have little, if any, effect on the CallPath Services API or the applications written to the API.

Collection of Call Detail Records

A **call detail record (CDR)** is a unit of information containing data about a completed telephone call, such as the time the call began, the date and duration of the call, the orig-



BCS = Business Communications System

TECH020-3

Figure 23. *CallPath/400 System Structure*

inating number, and the number that was dialed. Collecting call detail records is a CallPath/400 function that collects, validates and formats call detail records as received from a telephone switch. The **CDR collection manager** manages the collection of call detail information. An AS/400 application program can use the collected call detail records to generate management reports to assist in evaluating business operations, or to correlate specific business transactions to call activity.

A user may specify up to four collection objects on the AS/400 system for collecting call detail records. When starting the CDR collection manager, the user indicates to which of the following objects call detail records should be written:

CDR file member: CDRs that pass the validation tests are written to a file member specified by the user.

CDR data queue: CDRs that pass the validation tests are written to a data queue specified by the user.

CDR error file member: CDRs that do not pass the validation tests are written to a file member specified by the user.

Unformatted CDR data queue: CDRs received directly from the switch are written to a data queue specified by the user.

The format of call detail records received from different switch types are not identical. If collection to the CDR file member or the CDR data queue is specified, the CDR collection manager performs validation tests. It then formats the raw information as received from the switch into the specified objects in a consistent format. This shields the application from unique records for each switch type. Any records that do not pass the validation tests can be written to the CDR error file member. For users who prefer or require call detail records in the raw format from the switch, the collection manager writes the data to an unformatted CDR data queue.

Conclusion

The CallPath/400 program makes possible the development of AS/400 applications that integrate voice and data through the use of a functionally consistent application programming interface. Flexibility has been designed into the API so that it can be incorporated into new or existing applications, and so that it can be used with a number of supported switches.

The layered structure of the CallPath/400 program gives the AS/400 application program access to telephony functions provided by the switch through the API, yet shields the application from both the host's communications subsystem and the format and protocol details used by the switch.

The results may include enhanced application capability, improved customer service, and increased agent productivity.

Distributed Relational Database

Describes the AS/400 distributed relational database support, which allows users to access data on remote systems.

John M. Broich, Randy L. Egan,
Jeffrey W. Tenner, Carol L. Ramler,
Mark G. Wulf, and Teresa C. Kan

Introduction

Timely access to remote data is imperative in many businesses to manage for growth and global operations. Traditionally, access to remote data has been difficult due to complex coding for communications interfaces and transaction management. When the data is spread among a variety of computer systems, programming for data conversion makes access to remote data even more complex. The AS/400 system makes it easier to access distributed relational data by removing the necessity for complex coding of communications interfaces, transaction management, and data conversion.

The implementation of **distributed relational database** on the Operating System/400 (OS/400) database manager was designed to provide the same Structured Query Language (SQL) relational functions to remote systems as provided by local access to the database manager. The AS/400 database is an integrated database [1]. All database files on the system are managed by a single database manager and can be accessed by both relational and file model interfaces. This design allows for easier management of the database, less redundant data, and improved data integrity and usability. The database manager must coordinate distributed access with the **file model interfaces**, the traditional record-oriented access to database files. Distributed access through the SQL interface can be done concurrently with file access through distributed file support. For example, using SQL and distributed relational database, data from one system can be retrieved while data from another system is updated by distributed file support. However, the database manager ensures that distributed file support and distributed relational database run under the same unit of work (transaction).

Unique functions of OS/400 and SAA* Structured Query Language/400* (SQL/400*) licensed programs are supported when accessing remote systems. Some of these functions are:

- Interactive SQL support with statement prompting, validity checking, and column list aids
- HOLD clause on SQL COMMIT and ROLLBACK statements
- Single command for SQL program preparation
- Commit or rollback initiated by the host language or command language
- Commitment control lock level of *NONE
- Debugging and performance message support

By maintaining these functions, the application programmer does not have to relearn how to develop and test distributed applications.

A complexity that a distributed database manager must handle for an application is the transparency of data representations. A computer network consisting of unlike computer systems presents data in various representations. A programmable work station quite likely uses ASCII encoding for character data whereas an AS/400 system uses EBCDIC. Some computer systems store integer data in 2-byte words with the high-order byte last. Still others use different formats for floating point values. An application program should not be responsible for the various data representations and the rules for conversion. This function is included as part of the OS/400 database manager.

Maintenance of application code is expensive. Few organizations can afford to convert their applications to make use of a new function. With the SQL/400 implementation, SQL applications can become distributed applications merely by precompiling and compiling the program again.

Remote access requires additional processor resources and increases response time for database requests and data communications. The following design points minimize the effect on performance:

- Use bound queries created during program preparation
- Reduce the number of requests transmitted
- Minimize the amount of data transmitted
- Convert data representations only once

These are the design objectives of the OS/400 implementation of distributed relational database. This article discusses the architectures, components, and flows of this implementation.

Architectural Overview

The ability for the AS/400 system to access distributed data and for other remote systems to access AS/400 data is an integral part of the AS/400 system. Access between AS/400 systems (**like system environments**) and between AS/400 systems and other systems (**unlike system environments**) is achieved by implementing key architectures in the OS/400 program. These architectures include Distributed Relational Database Architecture* (DRDA*), Distributed Data Management (DDM) Architecture, Formatted Data Object Content Architecture (FD:OCA), and Character Data Representation Architecture (CDRA).

- Distributed Relational Database Architecture (DRDA)

  DRDA interchange flows and rules used for communicating between an application requester and an application server. The system where the application resides is known as the ***application requester***. The relational database manager where the data resides is known as the ***application server***. These flows and rules describe how the system performs the following functions:

  - Connect an application with a remote relational database. This entails allocation of a conversation between the systems using LU 6.2 and exchanging information between the systems, such as release levels, process coded character set identifier (CCSID) values, product type, and remote relational database name.
  - Bind an application with a remote relational database. A **bind** is the process where SQL statements and host variable information are sent to a remote relational database and converted to a control structure called a **package**.
  - Run bound SQL statements or dynamically prepared SQL statements on a remote relational database on behalf of an application and return any data and completion information to the application.
  - Maintain unit-of-work boundaries between the application and the remote relational database.
  - End the connection between the application and the remote relational database by deallocating the conversations between the application and the remote relational database.

The AS/400 system has implemented the first DRDA stage, called remote unit of work (RUOW). With RUOW, an application program running on one system can access data at a remote system within a unit of work. All SQL statements within the unit of work must be processed by the same database manager. However, an application can have a sequence of units of work where each unit of work is processed by a different database manager.

- Distributed Data Management (DDM) Architecture

  DDM architecture defines common interfaces for data interchange between systems. It is independent of a machine's operating system. A set of DDM data streams, constructed using architected commands, parameters, objects, and replies, defines the data interchange between like or unlike systems.

  The basic functions of DDM are:

  - Accepts requests, replies, and data from an application or a database manager, and constructs them into architected commands and objects.
  - Sends and receives DDM requests and replies from the communications facilities of the system.
  - Detects and processes normal and abnormal ending of communications.

- Formatted Data Object Content Architecture (FD:OCA)

  FD:OCA defines constructs that allow modeling of data or collections of data. The DRDA definition includes a special use of FD:OCA. A predefined descriptor object describes the mapping between DRDA data types[1] and FD:OCA representations. The mappings vary depending on the use of the data and system environment. Predefined descriptors also define common constructs that are composed of the DRDA data types. For example, a representation of the SQL communications area (SQLCA) has a predefined descriptor. A final descriptor, constructed when needed, describes data using the representations of the DRDA data types and common constructs. DDM defines an object for carrying FD:OCA descriptors and the data described by the descriptors.

- Coded Data Representative Architecture (CDRA)

  CDRA defines a set of CCSID values that uniquely identify the coded character representation used for character data. These CCSID values are used in FD:OCA defined constructs to tag data flowing between systems. When conversion of the data is necessary to preserve the value of the data, the system receiving the data performs the conversion.

Figure 24 on page 43 shows how the functional layers of the OS/400 operating system interact to perform distributed database processing described in this article. The main functional components are:

- The application containing embedded SQL statements, which access data at a remote relational database.
- SQL run-time support on the application requester, which interacts with the application program and interprets the SQL requests for the distributed database function of DDM.
- The distributed database function of DDM on the application requester, which converts the SQL operation into DDM commands.
- The DDM communications manager, which interfaces with the LU 6.2 implementation on the AS/400 system to send and receive DDM data streams. The DDM communications manager on the application requester is also responsible for managing the use of DDM conversations.
- The distributed database function of DDM on the application server, which converts DDM commands into SQL requests.
- SQL run-time support on the application server, which interacts with the local database manager to implement the database requests.

The following topics include a more detailed discussion of the creation of a distributed program, the connection to a remote relational database, and the processing that occurs when running a distributed application program.

[1] DRDA data types map closely to SQL data types.

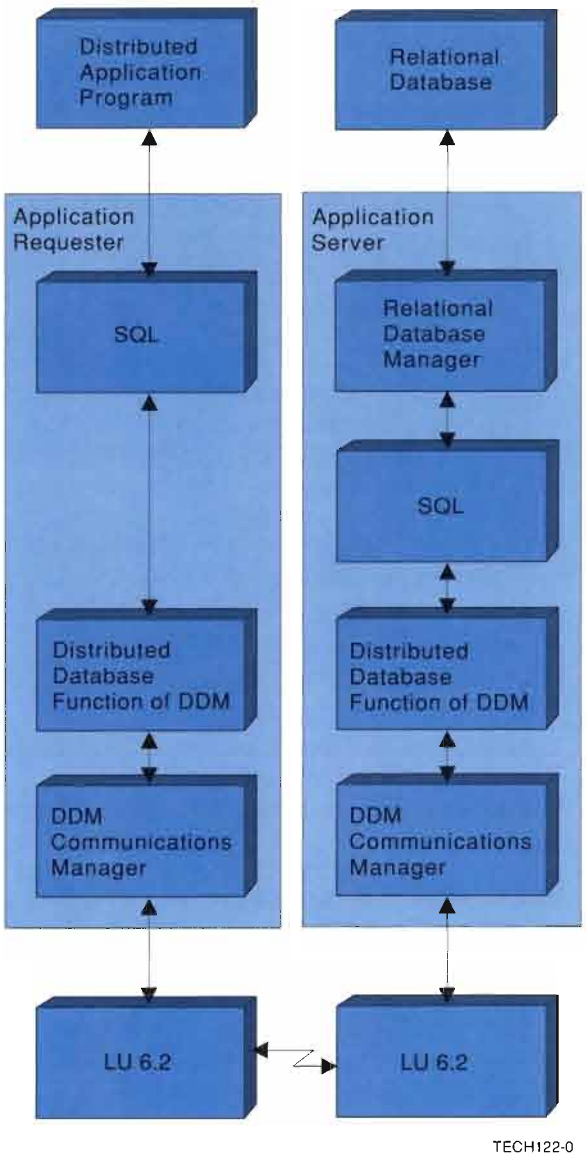

Distributed Application Program
Relational Database
Application Requester
Application Server
SQL
Relational Database Manager
SQL
Distributed Database Function of DDM
Distributed Database Function of DDM
DDM Communications Manager
DDM Communications Manager
LU 6.2
LU 6.2
TECH122-0

Figure 24. *Overview of OS/400 Distributed Database Support*

Creating a Distributed Application Program

Creating a distributed application program is similar to creating a nondistributed application program. The SQL precompiler commands were enhanced to allow the creation of a distributed application program. A distributed application program consists of two objects, the program on the application requester and the SQL package on the application server. An **SQL package** is a new OS/400 object used to support distributed database. It contains all the information necessary to run an SQL statement on the application server.

SQL packages are created during the DRDA bind phase. During the precompile and compile phases, all SQL statements are processed at the application requester. At this time the statement type and host variable attributes for each SQL statement in the program are determined and stored in the associated space of the program. A unique section number is assigned to each bound SQL statement. If the application requester does not recognize an SQL statement, a unique section number is still assigned. This provides the flexibility for the distributed application to contain SQL statements that are not recognized by the application requester but that the application server can run. A consistency token, which uniquely identifies this version of the program, is also generated.

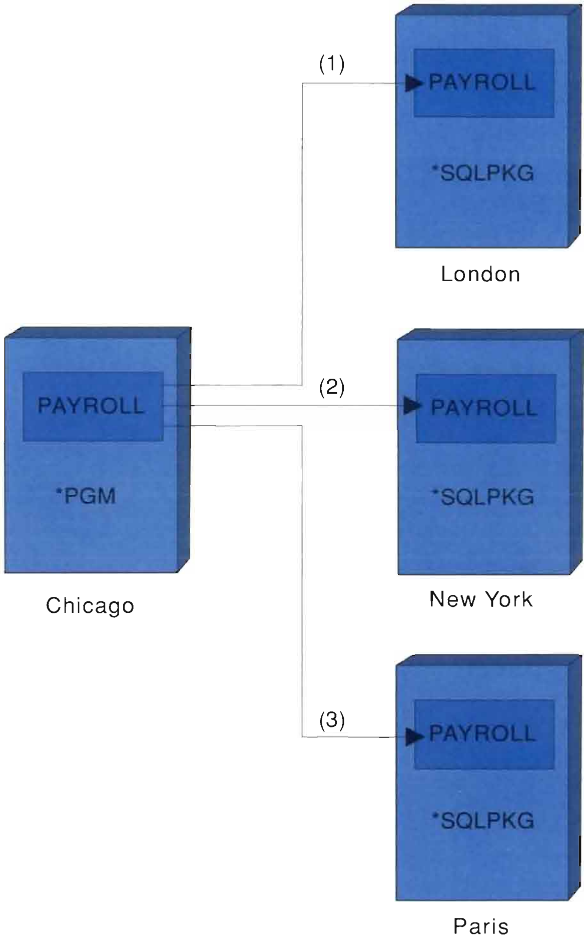
During the package creation phase, DRDA BIND flows are constructed from the information stored in the program-associated space and sent to the specified application server. If the application server is another AS/400 system, the DRDA BIND flows are extended to allow all the information about the SQL statements in the program to be sent in one flow to improve performance and reduce communications overhead. The application server processes the BIND flows and creates the SQL package. The precompile options, SQL statements, the CCSID of the SQL statement text, and host variable attributes are stored in the SQL package. The **access plans** (control structures used to process SQL statements when they are run) are built for the SQL statements and stored in the SQL package. Figure 26 on page 45 illustrates creating a distributed program. Specifying the RDB parameter on the Create SQL PL/I (CRTSQLPLI) precompile command indicates that a distributed program should be created. The program is created on the local system (CHICAGO), and then the SQL package is created on the remote system (ROME).

SQL packages enable the running of bound SQL statements on a remote relational database. Many implementations of distributed database processing only support dynamic running of statements. Dynamic running of statements has more run-time overhead than the running of bound statements because the SQL statements must be checked for syntax errors, and the access plans must be generated before the statement can be run. When running bound SQL statements, the overhead of syntax checking and generating access plans occurs only once, when the SQL package is created. With the AS/400 implementation of distributed database, bound statements can be run in a distributed environment, which improves run-time performance.

Because the information about each SQL statement is stored in the program, the package creation step may be repeated to distribute the SQL package to as many different application servers as necessary, using the program for input.

The following are advantages to using SQL packages to implement distributed database:

- There is only one object to manage at each system. Each application requester has a copy of the program, and each application server has a copy of the SQL package.
- Because the SQL package is created from the program and not the source, the program and the SQL package are synchronized.
- SQL packages may be created as needed and none of the application servers need to be in the network at program creation time. Figure 25 illustrates how SQL packages may be created on different relational databases in a network.



Note:

(1) CRTSQLCBL PGM(PAYROLL) RDB(LONDON)
(2) CRTSQLPKG PGM(PAYROLL) RDB(NEWYORK)
(3) CRTSQLPKG PGM(PAYROLL) RDB(PARIS)

TECH124-2

Figure 25. *Example of a Distributed SQL Application*

- The OS/400 save and restore commands can be used to save and restore SQL packages. An SQL package can be restored onto a different system than it was created on, and the first running of an SQL statement results in the access plan being rebuilt or an error if the appropriate files could not be found. This allows for the ease of distributing SQL packages in a like system environment.
- The OS/400 save and restore commands can be used to distribute a program to many application requesters that use the same SQL package on the application server.
- Because the SQL package is created from the program and not the source, the source no longer needs to exist on the system. This is beneficial to software vendors that do not want to distribute source. Also, the function to create SQL packages is shipped as part of the operating system, and the SQL/400 licensed program does not need to be installed on the system.
- Application development and testing may be done on a development system, and then the program may be distributed to the production systems without requiring that the source be distributed. The program can be installed and as many SQL packages can be created as necessary at the application servers.

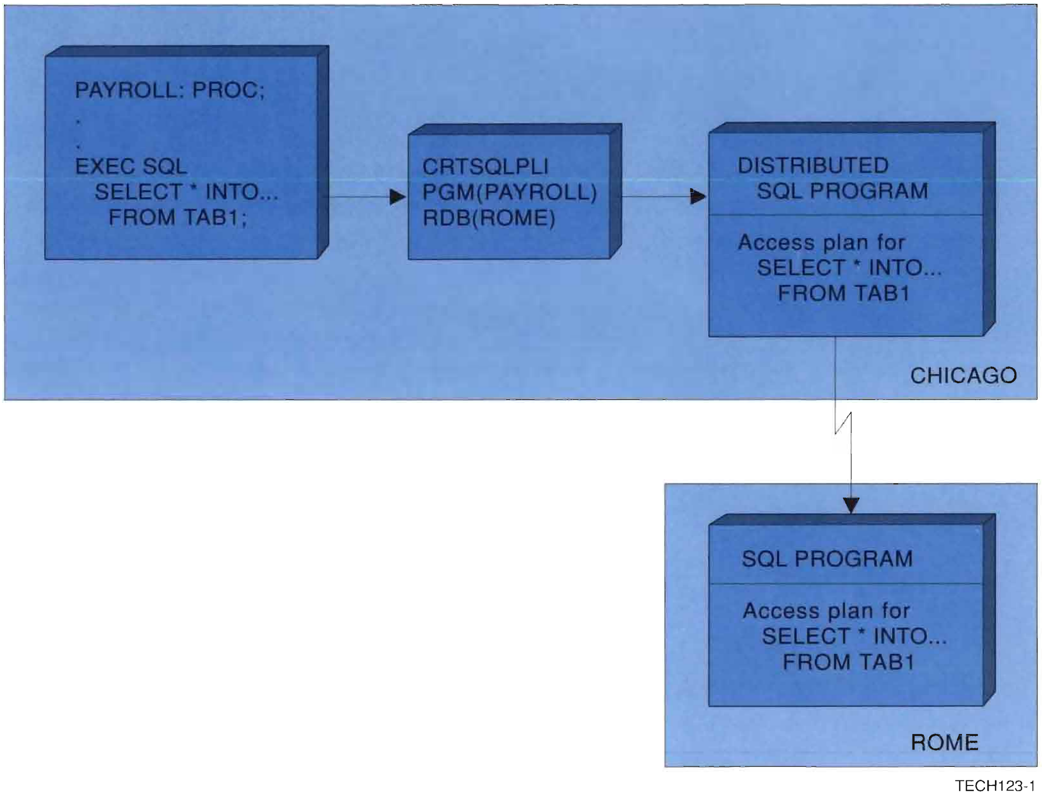


Figure 26. *Creating SQL Packages*

Connecting to a Remote Relational Database

The connection phase of a distributed database occurs when the application is starting its access to a remote relational database. This process involves starting a conversation with the remote system and then exchanging information to allow the application to communicate with the remote relational database. The information exchanged identifies the system type, architectural levels, default CCSIDs, and other characteristics of each system.

In designing and implementing distributed database connection processing, two goals were used in the decision-making process. The first goal was to minimize any changes to applications when converting from a local program to a distributed program, and the second goal was to share resources with the existing distributed file support (known as DDM on the AS/400 system). Sharing resources with DDM allows an application to use both DDM and distributed relational database within the same unit of work.

There are two types of connections that take place, implicit and explicit. Implicit connections occur when the first SQL statement in the distributed application program is run. The system connects the application to the relational database specified when the application was created through the SQL precompiler commands. An explicit connection takes place as a result of the SQL CONNECT statement. The SQL CONNECT statement identifies the relational database that the application wants to use as an application server. The SQL CONNECT statement can take the form CONNECT TO RDB1. This statement causes the application requester to disconnect from the current application server and connect to RDB1. Disconnection processing is discussed later.

For a connection to succeed, the application must not have a unit of work in progress. This is referred to as the connectable state. This state is defined to be either at the start of the application or at the end of a unit of work. A unit of work ends after a successful commit, which applies all of the operations in the unit of work to the database, or after a rollback, which removes the operations in the unit of work from the database. If the application is also using distributed files opened under commitment control, these files must be closed if the application is attempting to connect to a different remote system. When an SQL statement other then COMMIT, ROLLBACK, or CONNECT is run, the application is no longer in a connectable state and a connection cannot take place. If the application is not running under commitment control, an SQL CONNECT statement can be issued anytime because the application is always in a connectable state.

The application specifies where the data to be accessed is located by using the name of the relational database where the data is stored. This name is either specified on the RDB parameter of the SQL precompiler command or on the SQL CONNECT statement. SQL run-time support is able to use this name to retrieve the appropriate entry from the relational database directory. During a connection to a remote relational database, SQL run-time support determines if the application is in a connectable state and then gets the information necessary to complete the connection to the other system from the relational database directory. An entry in this directory contains a relational database name and the communications information needed to determine which communications device to use to communicate with the remote system.

The relational database directory entry is passed to the distributed database function of DDM. The distributed database function of DDM works with the DDM communications manager to start a conversation to the remote system and do the “handshaking” necessary to access the remote relational database. This handshaking includes sending two DDM commands, one built by each functional layer. These commands contain information, such as system type, default CCSIDs, and architectural levels supported. The system type and default CCSIDs help enable the automatic conversion of data between the two systems.

The distributed database function of DDM passes the relational database directory entry to the DDM communications manager, which controls the conversations used to communicate with other systems. The communications manager determines if a conversation already exists, which can be used to connect to the remote database. This determination is done by matching the communications information in the directory entry to the communications information of the existing conversations. If an existing conversation is not found, a new conversation is started using the communications information in the directory entry.

The DDM communications manager at the application server receives the commands sent by the application requester and, working with the distributed database function of DDM, builds replies that contain the same handshaking information sent by the application requester.

To get the information needed to build the replies to send to the application requester, the distributed database function of DDM passes the default CCSIDs sent from the application requester to SQL run-time support for validation. SQL run-time support starts commitment control at the application server and returns the CCSIDs of the application server to the distributed database function of DDM. The distributed database function of DDM returns these CCSIDs to the application requester. SQL run-time support at the application requester validates the CCSIDs from the application server, and the connection phase is completed.

The disconnection processing used in a like system environment is different than that used in an unlike system environment. Disconnection processing in an unlike system environment requires that the conversation to the remote system be ended to cause the disconnection. Because establishing the conversation is the most expensive part of connection processing, the AS/400 system does not end the conversation to the remote system to disconnect the application in the like system environment. The AS/400 system simply sends a disconnect command to the application server, which allows the application server to reset itself as if it were no longer connected. Resetting involves closing all cursors and destroying any prepared statements. The user can end the conversation at disconnection through an attribute of the application requester job.

Processing SQL Statements in a Distributed Application Program

Run-time processing of SQL statements includes processing of statements that are bound at the time the distributed application program is created. These include data definition language statements, data manipulation statements, and other miscellaneous SQL statements.

Dynamic statements are not bound when the distributed application program is created.

Instead, they are defined and run when the distributed application program is active. Some SQL statements use a cursor to read and manipulate data in the database. A cursor typically is used when multiple rows are to be read by the application program.

A more detailed discussion of the run-time processing of bound, dynamic, and multiple-row queries follows.

Processing Bound Statements in a Distributed Application Program

As stated earlier, the DRDA protocol minimizes the volume of data transmitted over the communications line while processing a distributed request. In the simplest case of processing a bound SQL statement without host variables, a 64-byte control structure is the only data sent from the application requester to the application server. This control structure identifies the SQL package and the section number in the SQL package that corresponds to the statement being run.

The SQL/400 licensed program supports the passing of data to and from the application program in host variables. Host variables are defined by statements in the host language (COBOL, RPG, C, FORTRAN, or PL/I) and are referenced in the SQL statements of the application program. In a distributed application program, input host variables from the application must be passed to the application server in addition to the control structure that identifies the statement to be processed.

When running a distributed application program in which the application requester is an AS/400 system and the application server is not, some host variable data types must be converted to those supported by the application server. An example of this is the binary-with-scale data type supported by the OS/400 database manager. This data type allows a number to be stored internally in a binary representation yet still have both a precision and scale associated with it. Because this data type is not supported by other database managers, it is converted to a decimal representation prior to being sent to the application server. Although this is not a DRDA-defined conversion, it is performed transparently in SQL run-time support to keep the programming interface for distributed application programs the same as that for nondistributed application programs.

Character string conversion is performed when necessary by the application server when character data is sent from the application requester. This conversion is performed based on the CCSID of the data being sent and how the data will be used on the application server. This conversion is done in accordance with CDRA guidelines.

It should be noted that the statements being processed in a distributed application program need not be recognized as valid statements on the application requester. The application requester sends the 64-byte control structure identifying the statement and any host variables for the statement to the application server. The application server, where the statement is valid, then processes it. This part of the DRDA protocol allows a distributed application program to use all of the SQL functionality available in the database manager at the application server.

Processing Dynamic SQL in a Distributed Application Program

Dynamic SQL allows an application program to define and run SQL statements while the application program is active. An example of an application using dynamic SQL is interactive SQL, an OS/400 application development tool that allows the user to enter an SQL statement on a command line and then run it. When processing dynamic SQL in a distributed application program, the SQL statement text and the CCSID of the statement text must be sent from the application requester to the application server. The application server converts the statement text to a different CCSID, if needed, prior to processing it. Using dynamic SQL, the CCSID of the statement text is set by SQL run-time support to the CCSID defined for the job unless specifically overridden by the application. In the case where the default is used, the application programmer need not be concerned with the CCSID for the statement text and need not be aware that conversion of the statement text may be taking place at the application server.

Processing Multiple-Row Queries

Some database queries return more than one row of data to the application. Application programs must use the following SQL cursor operations to process these queries:

- OPEN: The application program requests that the database query be started.

- FETCH: The application program requests that the values from the query's result table be placed in the host variables. The application program may change values or delete the row previously retrieved using a positioned UPDATE or DELETE.
- CLOSE: The application program indicates that no more rows are retrieved for this query.

Query processing is more complex than processing a single bound SQL statement because of the amount of application control of the process and the amount of data that needs to be returned. A discussion of the processing of each cursor operation follows.

Open Processing: Both the application server and the application requester have to prepare for the query. When an SQL OPEN statement is run, the following steps are performed:

1. The application requester determines if the cursor is already open. If so, an error is returned.
2. The DRDA open query request is transmitted to the application server including the values of any host variables used in the definition of the query.
3. The application server initiates the query. Two segments of information may be returned to the application requester:

   First, the description of the columns in the result table is returned. This is either a record format descriptor or an FD:OCA descriptor. A record format descriptor is always used when the application server and application requester are both AS/400 systems because it is the control structure used by the OS/400 database manager. The FD:OCA descriptor is a control structure that is understood by all DRDA participating systems.

   Second, the first block of result rows are returned to the application requester when the cursor is read-only and ALWCPYDTA(*OPTIMIZE) is specified as the precompile option.
4. The application requester receives the description and row data, and stores them so that they are available for fetch processing.
5. An SQLCA is returned to the application program indicating the completion of the OPEN request.

Fetch Processing: The application requester controls whether it has already received and stored a block of result rows, the position within the block, and the request for another block of result rows when needed.

The application server maintains the position in the database manager of where the next row or block of rows should be retrieved. When requested, it formats the data to be returned into either a query data block or an internal blocking structure.

The ***query data block*** is the DRDA control structure that contains the result row values of a query. If any value in the result row is the null value, only the null indicator is returned. Data of varying length character data types is returned with only the current length of the data. The rows of data in the query data block are of varying length. The alternative control structure is the internal blocking structure. This control structure consists of fixed length elements containing the result rows.

Each control structure has its own performance characteristics. The query data block allows the minimum number of bytes to be used to transmit null values and varying length character data. The internal blocking structure takes less processing time because it is the format used by the OS/400 database manager, so the application requester does not need to convert the query data block into the internal blocking structure format when running in a like system environment. In addition, the fixed length nature makes for easier addressability of the control structure.

For a like system environment, the internal blocking structure format is used unless a majority of the data of the result row is varying length character data.

For a like system environment where the program is created using the ALWCPYDTA(*OPTIMIZE) precompile option and for unlike system environments, the application server closes the query when it receives an end-of-data exception.

All conversions of data representations and data types for the result rows are performed by the OS/400 database manager on the application requester. The distributed application program need not be sensitive to the type of application server to which it is connected.

The application requester converts the values from either the internal blocking structure or the query data block into the host variable specified on the SQL FETCH statement. Because the application server has put the data into the control structure in the data representation of its choosing, the application requester must do the conversion of data representations as needed (for example, ASCII to EBCDIC). The CCSID of the result table columns, as identified in the FD:OCA or record format descriptor, is used to identify the character conversion necessary to the CCSID of the receiving host variables. The data type of the result data and the host variable may also be different (for example, the result might be an integer data type and the host variable a decimal data type). The application requester maps the value to the data type of the host variable. For values returned as null values, the host variable indicator variables are set appropriately.

Updating or deleting the current row is processed similar to any request to run an SQL statement except that the application requester checks that the cursor is open before sending the request to the application server.

Close Processing: If the query is closed by the application server during fetch processing, then all the application requester must do is return an SQLCA that indicates that the close request completed. If the application server does not close the query, the application requester must request that the application server close the query.

Queries that generate one block of result data are processed with only one request to the application server. During open processing, the first block of data may be retrieved and the query can be closed. The application requester receives this block of data with an indication that the query is closed on the application server. The application requester processes each row from the block without further communications to the application server.

The application server closes its access to the files of the queries similar to local processing.

Application Program Interface Considerations

An application program with embedded SQL statements calls SQL run-time support when running an SQL statement. The calling interface from the application program to SQL run-time support is the same for both distributed and nondistributed applications. This interface passes as parameters a structure (the SQLCA) used by SQL run-time support to return status information about the running of the statement to the application and any host variables that pertain to the statement. The SQLCA is updated by SQL run-time support after each SQL statement is processed to indicate whether the statement completed successfully. The SQLCA contains a field named SQLSTATE, which is set to a consistent value by all SQL application managers for a particular error condition. SQLSTATE gives the application programmer a single field to check for status without concern about whether the application is distributed or nondistributed.

SQL places messages in the job log whenever an error is encountered while running an SQL statement. These messages are based on information contained in the SQLCA. In an unlike system environment where the application server is not an AS/400 system, SQL run-time support attempts to map the SQLCA returned from the application server to an existing SQL message. In this way, the message help and message variables are available to the distributed application developer regardless of the environment in which the distributed application program is running.

Commitment Control in a Distributed Application Program

Commitment control is a means of grouping database file operations that allow the processing or removal of a group of database changes as a single unit. Commitment control support under RUOW requires that all SQL statements within a unit of work be processed at a single relational database. The database manager on an AS/400 system ensures that every request is processed at one and only one system within a unit of work.

The lock levels supported for distributed and nondistributed database processes are the same. Four lock levels—*NONE, *CHG, *CS, and *ALL—are supported by the OS/400 database manager. In a distributed environment, the lock level *NONE is only supported between AS/400 systems. Although the repeatable-read lock level is not supported on the AS/400 system, an exclusive lock level is acquired on the AS/400 application server when a lock level of repeatable read is requested in an unlike system environment.

Commitment control support on an AS/400 system allows both file (using DDM file) and database (using SQL) accesses to run concurrently on the same remote system. Managing a unit of work that contains both distributed file access and distributed database access is more complicated than managing a single access within an application. All remote database objects and files within an application must be closed and committed or rolled back before an application connects to another system. However, the connection restrictions do not apply to files and database applications that are not running under commitment control.

While compiling or running an SQL program, commitment control is implicitly started by SQL. The lock level used when running an SQL program is determined by the level specified on an SQL precompiler option. When a distributed DDM file is placed under commitment control, the commitment control requirement is the same as that for a local file running under commitment control. For DDM, the user must explicitly start commitment control by issuing a Start Commitment Control (STRCMTCTL) command with the specific lock level. Then, the distributed DDM file can be opened under commitment control.

Changes to resources (for example, file requests and database requests) running under commitment control on the OS/400 program are tracked under an internal object called a commitment definition. When the first request is performed against a remote resource, the resource is added to the commitment definition during the connection process. Communications information and the resource name, which is either a relational database name or a DDM file name, are stored in the commitment definition. All subsequent requests performed against remote resources must have the same communications information because only one conversation is allowed to be established at any time. Only one remote relational database can be added to the commitment definition since only one relational database is allowed per system. On the other hand, multiple DDM files with the same communications information can be added to the commitment definition. The remote resources may be removed from the commitment definition at a unit-of-work boundary.

When a commit or rollback of a unit of work is performed, the database manager extracts the communications information from the commit resource directory. The database manager uses this information to identify the application server and then sends the request to perform the appropriate function. The commit and rollback functions apply to all resources within the conversation on an AS/400 system. All SQL cursors are closed unless the HOLD clause is specified on the SQL COMMIT or SQL ROLLBACK statement. In an unlike system environment, the cursors are closed whenever the commit function fails. DDM files are not closed after a commit or rollback request is performed. An explicit close request of the DDM file must be issued to bring an application into a connectable state. The conversation is deallocated when a DRDA process is ended or when a ROLLBACK function fails.

Commitment control ends when a job ends (either normally or abnormally), or the user issues the End Commitment Control (ENDCMTCTL) command. An INQUIRY message is issued if end commitment control is requested and changes are pending. The user has a choice to commit or rollback the changes before ending commitment control or to cancel the request. The conversation is deallocated when commitment control is ended on the remote system.

Conclusion

Transaction management, communications, and other complicated aspects of a distributed application are performed by OS/400 components to make developing and maintaining distributed applications easier than in the past. Because the SQL interface is essentially the same for distributed and nondistributed applications, programmers familiar with SQL are able to develop distributed applications without a prolonged learning period. Distributed database support coupled with the OS/400 database provides customers with the tools to develop and put into production distributed applications without sacrificing relational functionality.

Reference

1. Anderson, M.J. and Cole, R.L., “An Integrated Data Base,” *IBM Application System/400 Technology*, SA21-9540, 20–24. June, 1988.

Cooperative Processing

Describes how personal computers and AS/400 systems work together to process data.

David A. Wall, Phillip C. Schloss,
Janet H. Krueger, Clark A. Scholten,
Patrick T. Priniski, and Kathryn D. Cook

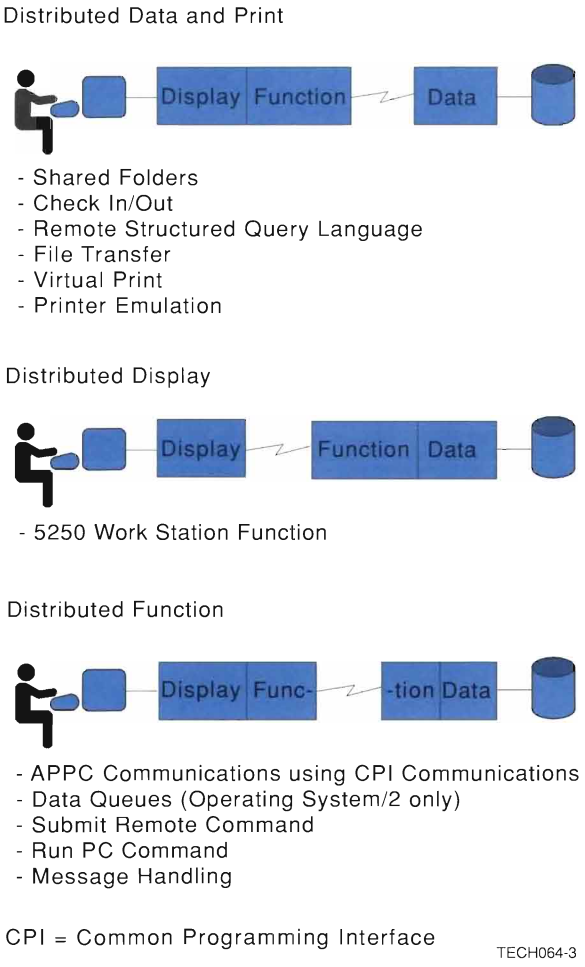
Introduction

Cooperative processing is the handling of data where two processors are involved: a programmable work station (PWS) and a host. The application developer splits the function into two pieces to run on the PWS and the host. This article describes how PC Support participates in cooperative processing. The article describes:

- How PC Support application programming interfaces (APIs) are used to create cooperative processing applications
- How the PC Support router ties the AS/400 system to the personal computer
- The PC Support data queue function, which provides direct access to AS/400 data queues
- The PC Support remote structured query language (SQL) function, which provides direct access to AS/400 database files

Cooperative Processing with PC Support APIs

Cooperative processing combines the personal computer and AS/400 system environments to support advanced functions and applications. PC Support offers functions to support cooperative processing and has an organizer function that provides an integrated end-user view of PWS and host applications. Using the functions of PC Support, customers and business partners can write applications that run both on the host and the personal computer. Figure 27 shows how the function of a program can be divided between the personal computer and the host. PC Support has functions to support all of these environments.



- Shared Folders
- Check In/Out
- Remote Structured Query Language
- File Transfer
- Virtual Print
- Printer Emulation

- 5250 Work Station Function

- APPC Communications using CPI Communications
- Data Queues (Operating System/2 only)
- Submit Remote Command
- Run PC Command
- Message Handling

CPI = Common Programming Interface

Figure 27. *Distributed Processing*

PC Support provides APIs to help integrate PC Support functions with the user's application. The user can code directly to the API because a description of the APIs is in the PC Support publications. PC Support also provides a high-level interface to the API calls in the form of function libraries. These libraries are available for many of the personal computer programming languages. The user can link the library function directly into their application. This lets the user make a simple call to the PC Support library instead of making the complex API call to perform the function.

The APIs are available so the user can integrate PC Support function directly into their application. The user can use PC Support functions to access the data and write their own application to use the data. The application can be on the personal computer or on the AS/400 system. For example, the user can use the router API to move data between the personal computer and the AS/400 system. Using the router API gives the user an easy way to transport the data. This lets the user concentrate on the details of processing the data instead of writing complicated communications code.

PC Support also provides tools and examples to help the user write cooperative processing applications. This information is provided in an AS/400 system folder that is optionally installed when PC Support is installed on the AS/400 system. Example programs and include and .H files are in the tools folder. The include and .H files are the library functions provided for each PC Support API. The example programs show API use. Some of the programs show what is needed to use the basic function. Others are small applications that show possible uses for an entire group of API calls. The examples are not intended to be used as provided but to help users understand how to incorporate the APIs into their applications.

The tools folder also contains programs that can run on personal computers and batch files. These programs and files are provided to help users create hypertext files and help information for their applications.

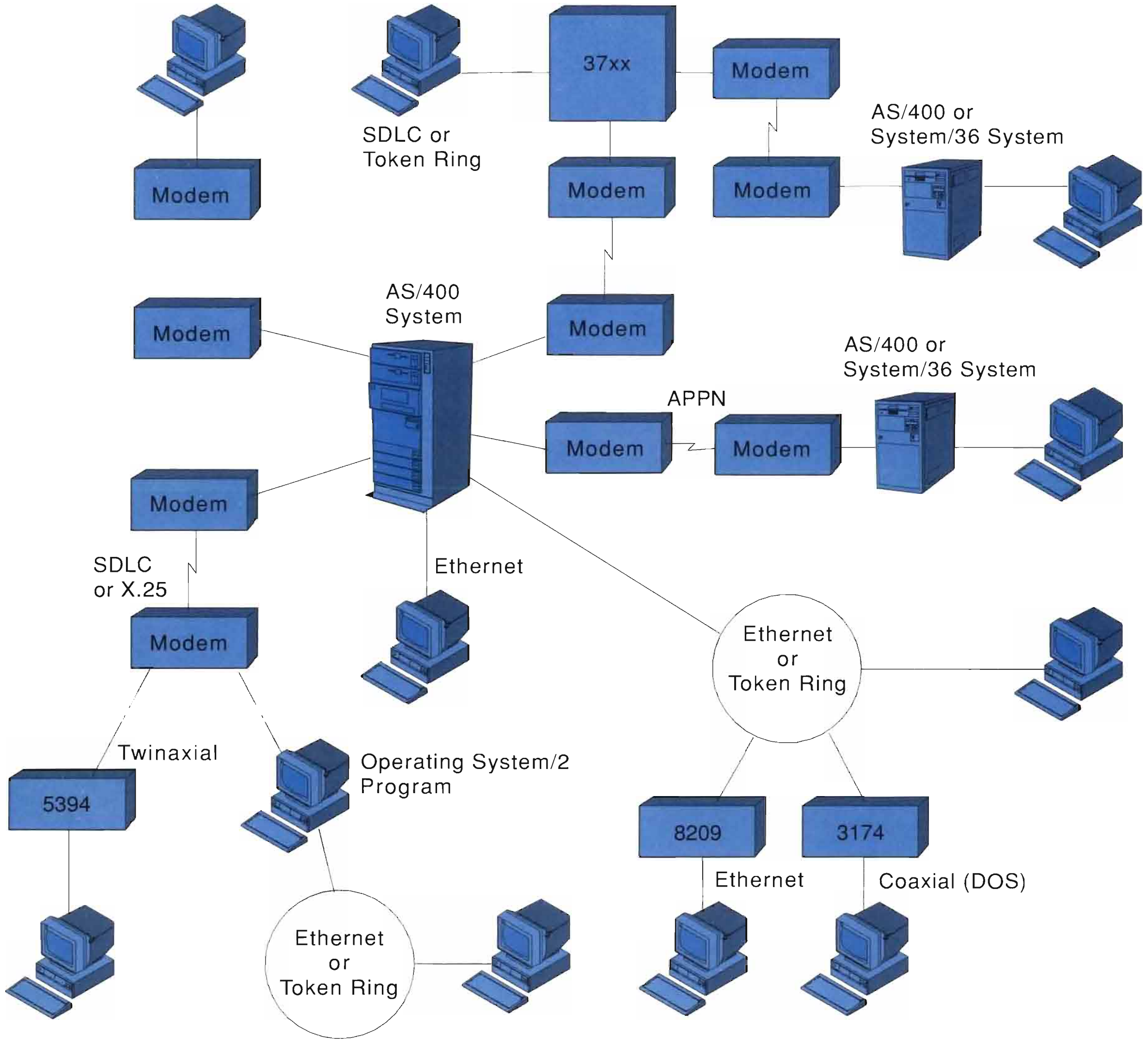
PC Support/400 Router

The foundation for cooperative processing is the PC Support/400 router, which provides connection services between the personal computer and the AS/400 system. The services provided follow the advanced program-to-program communications (APPC) or logical unit 6.2 (LU 6.2) communications architecture. Numerous connectivity options are available as shown in Figure 28 on page 54.

PC Support provides a wide range of connection services for both local and remote users. Local area network (LAN) users can attach to the AS/400 system through the IBM token-ring LAN or Ethernet LAN. The LAN connection uses integrated adapters on the AS/400 system. Personal computers can also be attached to the AS/400 system through a twinaxial or asynchronous connection. Coaxial devices are supported through a protocol converter or device controller. Remote locations are supported over public networks, switched and leased. Remote concentrators, such as the IBM 5394 Remote Control Unit, allow work groups of personal computer users at a remote site.

The structure of the PC Support/400 router allows the application to be independent of the connectivity. Cooperative processing applications that use the router become LU 6.2 applications. This allows each work station user to access resources anywhere in an APPN network, no matter how the work station is attached.

For environments where performance is critical, the 16Mbps token-ring attachment is recommended. The maximum frame size allowed for data transfer in the token-ring router increased from 8KB (KB equals 1,024 bytes) to 16KB. The larger frame size improves the performance of the token-ring router and reduces the processing overhead needed to transmit and receive large amounts of data between a personal computer and the AS/400 system on a 16Mbps token-ring network.



APPN = Advanced Peer-to-Peer Networking
SDLC = Synchronous Data Link Control

TECH060-4

Figure 28. *PC Support Connectivity*

Data Queues

A personal computer user now has the ability to interact with AS/400 data queues through a programming interface. A **data queue** is an AS/400 object that provides a method of interprocess communications. That is, one process sends data to another process through a data queue. Present on the system since Version 1 Release 1, data queues allowed fast, efficient communications between AS/400 programs. Beginning with Version 2 Release 1, this function is available to the personal computer. PC Support added support for data queues in the form of an application programming interface (API). All data queue functions available to the AS/400 programmer are extended to the personal computer programmer.

Data queues can be accessed by multiple requesters and servers. The requester or server may be located on the personal computer or the AS/400 system. The **requester** places free-form data on the data queue. This data is retrieved and processed by the **server**. In turn, the server places its response on a data queue that any of its requesters can later retrieve. Either the requester or the server can be a PC program. Figure 29 on page 55 shows how data queues can be used.

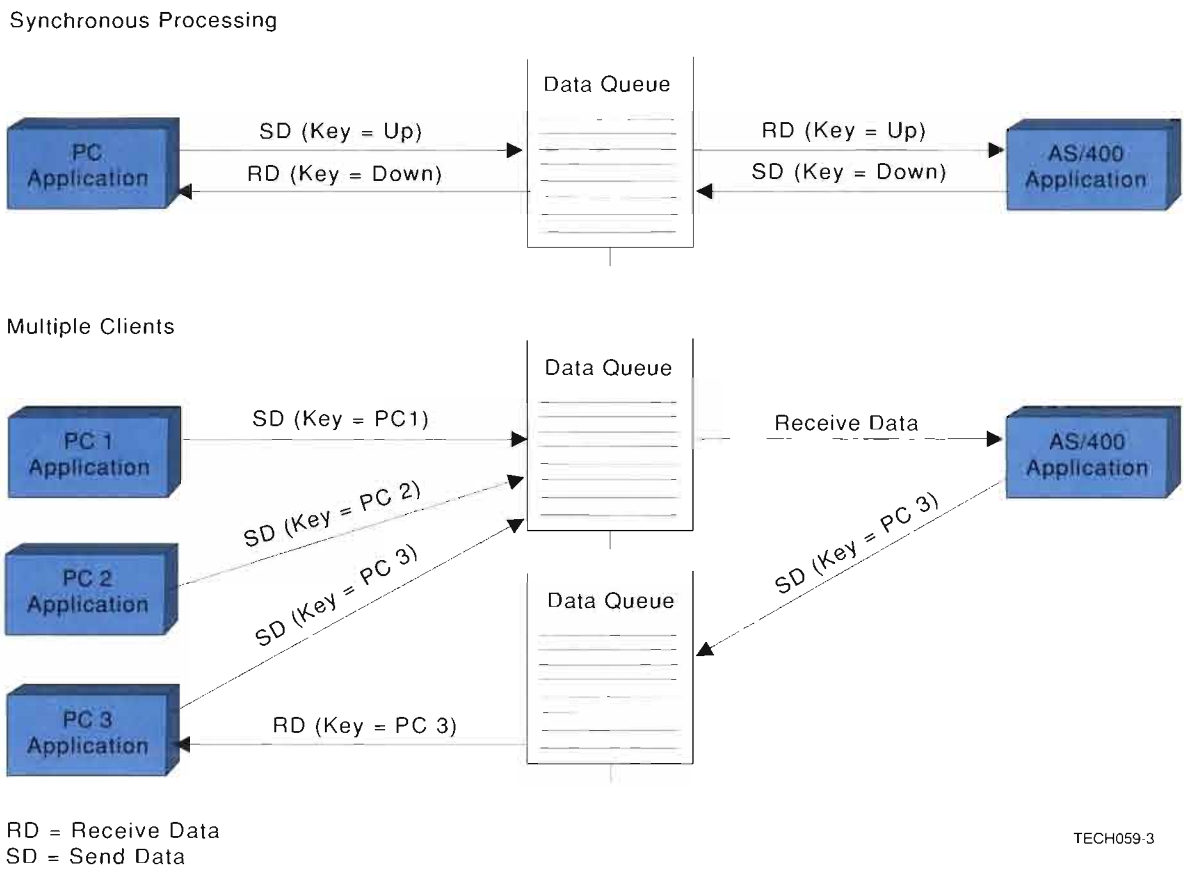


Figure 29. *PC Support Data Queues*

Data sent to the data queue is referred to as a message. The maximum length of a message for a data queue is determined when the data queue is created. The personal computer can send up to 31KB of information in a single message. Ordering the messages is done in one of three ways. The first message sent to the data queue is the first message retrieved; this is also known as first-in first-out (FIFO). Similarly, last-in first-out (LIFO) has the last message sent to the data queue being the first message retrieved. Messages can also be arranged according to a key that is sent with the message; this is referred to as a keyed data queue.

The PC Support implementation of data queues is transparent to the server. This means existing AS/400 programs that use data queues can have any part of their function moved to the personal computer. For example, the end user interface of a data entry program can be on the personal computer. This makes the graphical user interface and other ease-of-use features of the personal computer available to the application.

Users writing applications for the Operating System/2 (OS/2) licensed program can use data queues through an OS/2 dynamic link library (EHNDQAPI.DLL). The data queue's DLL is shipped as part of PC Support. The following actions can be performed on an AS/400 data queue from the personal computer:

- Creating a data queue
- Sending messages to a data queue
- Retrieving messages from a data queue
- Deleting all messages from a data queue
- Querying the attributes of a data queue
- Deleting a data queue

In addition, the following options are available to the personal computer user:

- Retrieving a message from the AS/400 system if one of the actions fails
- Ending the communications link with the AS/400 system

PC Support provides an import library (EHNDQ.LIB) so users can easily link data queue functions to any high-level language. Example programs that use data queues are available in the tools folder.

PC Support added the remote Structured Query Language (SQL) function in Version 2 Release 1 of the AS/400 system to provide an easy, convenient way for PC-based application programs to run SQL statements on an AS/400 system. Remote SQL provides an application program interface (API) that allows personal computer applications to access SQL database files and other database files on the AS/400 system.

For both the disk operating system (DOS) and OS/2 environments, the API is accessed with a small library that is linked with the application program. The user's program communicates with the remote SQL program that also resides on the personal computer. The remote SQL program communicates with an AS/400 program to run the SQL functions.

Remote SQL provides the following APIs to perform SQL functions on the AS/400 system.

- SELECT rows
- FETCH rows that satisfy a given SELECT
- UPDATE the row at the current cursor position
- DELETE the row at the current cursor position
- Other SQL functions by calling the Execute Remote SQL API

Additionally, a PC application can use the remote SQL function to provide basic program-to-program communications with an AS/400 application. Remote SQL provides the following functions through an API:

- A PC application can call an AS/400 application

- An AS/400 application can call an OS/2 application

- Cooperative processing applications running on an AS/400 system and on a personal computer can exchange character data

Conclusion

Achieving cooperative processing between personal computers and the AS/400 system is possible using the cooperative processing functions of PC Support/400. PC Support provides the functions users need to write cooperative processing programs. Users can write programs using the PC Support APIs so personal computers can share data with the AS/400 system, which expands the power of both the personal computer and the AS/400 system.

Advancements in PC Support/400

Describes the changes made to PC Support since Version 1 Release 1.

David A. Wall, Mark G. Wenzel,
Timothy L. Kramer, Janice R. Glowacki,
Janet H. Krueger, and Ann M. Bukowski

Introduction

Functions have been added to PC Support/400 (known as AS/400 PC Support at Version 1) since its initial release in Version 1 Release 1. The existing functions of PC Support have also improved in this time. These changes improve the usability, performance, and function of personal computers attached to the AS/400 system. This article describes the following enhancements to PC Support:

- The IBM disk operating system (DOS) version of PC Support takes advantage of new technology to increase the amount of conventional memory available to personal computer (PC) applications.
- PC Support/400 installation and configuration is now easier to use and provides support for a central administrator.
- The performance of PC Support/400 shared folders, the PC Support function providing transparent file serving, is improved.
- The AS/400 system can participate in a network with PC servers. PC Support/400 can coexist with both Novell** and OS/2 servers.
- Information on PC Support is available online, with all topics interconnected by hypertext links.

Memory Reduction

The original IBM Personal Computers used the Intel** Corporation 8088 processor, which had maximum addressable space of 1MB (MB equals 1,048,576 bytes). The DOS operating system, developed for the IBM Personal Computers, was designed to operate within this 1MB addressable space. The upper 384KB (KB equals 1,024 bytes) of this space, which was reserved for personal computer adapters and basic input/output services (BIOS), became known as the **reserved adapter area**. The remaining 640KB of the addressable 1MB area was intended for the DOS operating system and its applications and is known as **conventional memory**. See Figure 30 for an illustration of DOS memory.

As applications become more sophisticated their memory requirements grow. New techniques have been developed to simulate multitasking under DOS that further stress the memory requirements. This memory growth results in problems where users cannot run all of the functions they want in the personal computer at the same time. Thus, conventional memory has become a limited resource in the personal computer and has to be managed carefully by both the user and the applications.

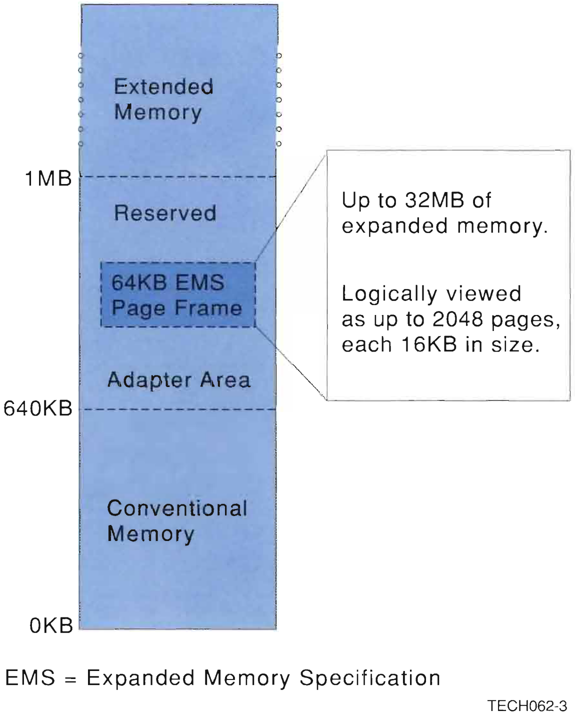

Extended Memory
1MB
Reserved
64KB EMS Page Frame
Adapter Area
640KB
Conventional Memory
0KB
Up to 32MB of expanded memory.
Logically viewed as up to 2048 pages, each 16KB in size.
EMS = Expanded Memory Specification
TECH062-3

Figure 30. *DOS Memory Map*

Enhancements have been made in PC Support since Version 1 Release 1 to minimize the amount of conventional memory PC Support functions use. This minimizing of conventional memory is accomplished by:

- Using the Expanded Memory Specification (EMS)
- Providing a function to remove PC Support functions
- Using extended memory

Expanded Memory Specification

The Expanded Memory Specification (EMS) was developed by the Lotus Development Corporation, Intel Corporation, and Microsoft Corporation. This specification describes a way in which an application can address more than 640KB of memory by using **expanded memory**. The personal computer can address more memory by defining an area known as an EMS page frame. The **EMS page frame** acts like a window where memory is moved in and out of the personal computer's addressable area. This is illustrated in Figure 30 on page 58. The EMS page frame is normally defined to be a 64KB area within the reserved adapter area. The EMS page frame is further divided into 16KB regions known as physical pages.

The EMS programs allocate logical pages of expanded memory of 16KB, which are moved in and out of the EMS-page-frame area as needed. Thus, applications can be developed to use EMS, which page the proper area of memory in and out of the EMS page frame as needed. Because the pages are mapped in and out of the page area as needed by the application, instead of all the pages being defined in the 1MB addressable space, the amount of conventional memory used by these applications is reduced.

PC Support uses EMS through the PC Support/400 memory manager program EIMPCS.SYS. EIMPCS.SYS is a DOS device driver. It provides a generic interface for the PC Support functions that make memory requests. EIMPCS.SYS handles the memory requests by performing the appropriate EMS or DOS memory request on behalf of the application. Because EIMPCS.SYS makes all memory requests, it can make requests to EMS, if available, before using conventional memory.

The centralization of memory functions in EIMPCS.SYS frees the PC Support applications from handling the details of the EMS or DOS memory environments. EIMPCS.SYS does the appropriate conversions and requests. This design requires that only EIMPCS.SYS change if some techniques or enhancements concerning EMS or DOS memory occur in the future.

Removal of PC Support Functions

The remove PC Support functions program (RMVPCS.EXE) was developed for Version 1 Release 2. This PC Support program allows users to remove resident PC Support functions from memory. When PC Support functions are removed, more conventional memory is made available for other applications.

RMVPCS.EXE stops a PC Support function before the function is removed. The memory used by the PC Support function that was removed is now available for use by other personal computer applications. The PC Support function can be reloaded into memory if it is needed at a later time.

Extended Memory

As technology improves in the computer industry, personal computers are using the more advanced Intel Corporation processors. These processors include the 80286, 80386, and 80486. These advanced processors all have a mode that emulates the earlier Intel Corporation 8088 and 8086 processors. This mode is known as **real mode**. In this mode these processors retain the limitation of the 1MB addressable space. These processors also have a more advanced mode known as **protected mode**. This mode allows these processors to address more than 1MB of memory. The memory addressed from 1MB and above is known as **extended memory**.

Even though the advanced processors are capable of addressing more than 1MB in protected mode, the DOS operating system was intended to run in real mode. Thus, extended memory is not normally available to DOS applications.

DOS extenders are programs that allow DOS applications to run in protected mode. Applications that use DOS extenders to run in protected mode are then able to use extended memory. The DOS extender handles switching the processor between real and protected modes at the appropriate times for such things as hardware services, DOS requests, and BIOS requests. The DOS extender also is responsible for the management of extended memory.

The PC Support/400 extender interface program (PCSXI.EXE) provides PC Support programs with a general interface to specific DOS extender functions, much like EIMPCS.SYS provides a general interface for expanded memory management. Again, having all the PC Support functions make requests to PCSXI.EXE eliminates the problem of having duplicate DOS extender code in memory. This in turn conserves even more conventional memory. Also, the flexibility for enhancements or the use of multiple DOS extenders is in one central component.

Administration Program

As the number of personal computers using PC Support/400 to access the resources of the AS/400 system increases, the need for central control of those personal computers also increases. In the past, configuration control of a large number of personal computers was difficult because the configuration files actually reside on the personal computer. Administrators who wanted common configurations across a group of users or who made changes to individual user configurations had to make the change at that user's personal computer or send each user a diskette. The PC Support administration program addresses this problem.

The PC Support administration program lets a single user update the configuration files of PC Support users from a single location. The administrator creates or changes the configuration files for the PC Support users in folders on the AS/400 system. When PC Support is started by the user, the PC Support update function copies the new and changed files from the AS/400 system to the end user's personal computer.

Administrators can easily create and change configuration files for many different users without leaving their own personal computers. Even remote users located in a different city or state can receive configuration changes from the administrator.

The administrator can also create customized installation diskettes for new PC Support users. After the administrator supplies all the information necessary to install PC Support on the user's personal computer, the end user simply inserts the diskettes and types INSTALL.

Configuration Folders and Administration Authority

The administration program manages configuration information on the AS/400 system in folders. Each set of configuration information the administrator manages is in its own folder. The PC Support installation program creates the folder QIWSADM and its subdirectories MODEL and USER. Then, depending on the type of configuration being administered, the administration program creates the subdirectory I:\QIWSADM\MODEL or I:\QIWSADM\USER.

PC Support administrators have *ALL authority to the various folders maintained by the administration program (QIWSADM and its subdirectories). Therefore, not all users have administration authority. A user is an administrator only when defined as such on the AS/400 system.

Model and User Configurations

The PC Support administration program is used by the administrator to control the configurations of each PC Support user. The administrator can work with model configurations. Model configurations store configuration options that are common for a certain group of PC Support users. For example, model configurations may be created for different departments or different types of users (such as secretaries, managers, or programmers). Model configurations contain information about resources that are common to the group. For example, common printers or common folders could be included in a model configuration.

After the necessary model configurations are set up, the administrator creates user configurations for each individual PC Support user. Like model configurations, a user configuration is stored in an AS/400 folder. User configurations are often created by copying an existing model or user configuration. When this is done, all the files that exist in the existing configuration are copied to the new user configuration folder.

Each user configuration contains various configuration files for the individual PC Support user. For example, these files include the PC Support command file and the PC Support configuration file. In addition, if the user configuration was set up for the work station function, the work station function master profile, the associated session and keyboard profiles, printer function tables, and the session manager profile might be included as configuration files.

The user configuration must be complete. That is, the configuration must contain the needed router information to establish a connection to an AS/400 system. For example, the configuration must contain the name of the personal computer and the specifics about the way the personal computer is connected to the AS/400 system. Administrators supply the required router information when they create a user configuration.

Configuration Updates from the Administrator

PC Support users do not need to take any special action to receive updated configuration files from the administrator. Configuration files that have been added or changed by the administrator are automatically copied to the user's personal computer when the user starts PC Support.

PC Support users who receive updates from the administrator have an identifier in their PC Support configuration file. This identifier defines the source and destination of the configuration files controlled by the administrator. The **source** is the user configuration on the AS/400 system where the master copy of the user's configuration files are stored (for example, I:\QIWSADM\USER\JOE). The **target** is the directory on the user's personal computer where the files are copied (for example, C:\PCS). The PC Support/400 update function, which is called each time a user starts PC Support, copies any new or changed files from the source location on the AS/400 system to the target location on the user's personal computer.

Customized Installation Diskettes

The administrator can also create customized installation diskettes. At Version 1 of PC Support, users who installed PC Support on their personal computer, using the standard installation diskettes, needed to understand various configuration options and hardware connectivity information. This caused frustration for the user who did not have this knowledge. The administration program solves this problem because it lets a PC Support administrator create customized installation diskettes, which require no interaction from the end user.

When the administrator creates user configurations, the information in the PC Support working set files have complete connectivity and selected functional configurations. Then, when creating a customized installation diskette, the working set files, along with the necessary installation, router, and shared folder programs are copied to the customized installation diskette. In this way, the user can start the installation program from the customized diskette and simply wait for the installation to complete.

Shared Folder Function

The PC Support/400 shared folder function provides transparent PC file serving. That is, PC files stored on an AS/400 system are accessed by the personal computer as if they were stored locally on the personal computer. The shared folder function has been enhanced since Version 1 Release 1 to perform better and to run with the Operating System/2 (OS/2) licensed program. Performance is improved through better use of the data cache. These enhancements are discussed in the following subtopics.

Cache Enhancements

The shared folder function uses a data cache to reduce network traffic communicating to the AS/400 system. This improves the overall capacity of the AS/400 system and improves the PC user's response time by processing data locally (on the personal computer) rather than remotely (on the AS/400 system).

Every AS/400 system access takes a significant amount of time. The use of a data cache helps performance because the shared folder function does not have to access the AS/400 system as often to send or get data. That is, fewer large data requests are more efficient and faster than more small requests.

The shared folder function used a data cache in Version 1 Release 1, but the algorithms to use the cache have improved. The following paragraphs describe these improvements.

A **cache** is a data buffer that temporarily holds user data. The shared folder function uses the data cache during read requests by getting more data from the AS/400 system than the application requests. Performance improves if the application is reading data from the file in approximately the same place in the file. On the first read request, a large amount of data is retrieved from the AS/400 system and placed in the cache. On subsequent read requests, the data in the cache is used instead of going to the AS/400 system to get the data.

The shared folder function uses the data cache during write requests by storing the data in the cache until the cache is full or the data needs to be sent (for example, the data is sent to the AS/400 system when the file is closed). Performance improves if either of the following happen.

- If the write requests overlap (that is, if some of the data is written to the same place in the file), then the data in the cache is updated. Only the updated cache is sent to the AS/400 system so that the AS/400 system handles only one request.
- If the write requests are sequential (that is, if they are to consecutive places in the file), then the write requests are combined in the cache. One large request is sent to the AS/400 system instead of several small ones.

Placing the data in the cache also helps performance if the application is writing data that is not changed. The shared folder function recognizes that the data has not changed and does not send the data to the AS/400 system.

Using the data cache for read and write requests is achieved through a paging algorithm. That is, memory used for the cache is divided into pages. A **page** is an independent unit of the cache that can hold 1KB, 2KB, or 4KB of user data. Each file that has data in the cache has the data in pages. The data is kept in these pages for a limited period of time. The duration of the data in the cache is determined by a least-recently-used (LRU), scheme. If a cache page is needed and there are no available pages, then the page that was used the earliest is made available. In general, a larger cache means better performance because there are more cache pages to hold data. For DOS, conventional memory from the 640KB area, expanded memory, or extended memory may be used for the cache.

Knowing what personal computer applications often do helps the shared folder function reduce the number of pages sent to the AS/400 system. Personal computer applications usually read some amount of data, change part of the data, then write the data back to the file. Some or all of the data that was read is not changed as it is written back to the file. Much of the data has not changed; therefore, sending the data to the AS/400 system creates unnecessary network traffic. The shared folder function does not send unchanged pages of the cache, resulting in less data being sent to the AS/400 system.

Another common practice used by applications is to open a file, read data from it or write data to it, close the file, then reopen the file, and read the same data. The shared folder function uses the data cache to improve the performance of this type of application by saving the pages for a closed file. When a file is closed, information about that file and its cache pages are kept in the cache. They are kept in the cache until the file is reopened or until cache pages are needed (the pages of a closed file are the first pages made available if cache pages are needed). When the file is reopened, the shared folder function checks the cache to determine if pages for that file are still in the cache. If pages exist for that file and the file has not changed since it was closed, then the pages contain valid data for the file. The shared folder function discards data in the cache if the file has changed.

When an application reads or writes data, the cache is successful if the shared folder function does not have to access the AS/400 system to complete the operation. For example, if the data for a read request is already in the cache, the data in the cache is used for the request. The cache was successfully accessed. The shared folder function keeps track of the number of times the cache is successfully used. An application in the PC Support tools folder retrieves this information. The user can use this information to tune the cache. Statistics can be retrieved on a larger or smaller cache to determine how the size of the cache affects the performance of the cache.

OS/2 Support

The shared folder function is available under both DOS and OS/2 personal computer operating systems. The shared folder function provides support on the OS/2 operating system through an OS/2 file system driver (FSD) that follows the OS/2 installable file system (IFS) interface. The shared folder function receives requests from the OS/2 operating system through its FSD. The shared folder function FSD runs at OS/2 privilege level 0. It passes the request to the part of the shared folder function that runs as an OS/2 application at OS/2 privilege level 3. The request is formatted into a distributed data management (DDM) data stream and is sent to the AS/400 system by the OS/2 communications manager. This flow is shown in Figure 31.

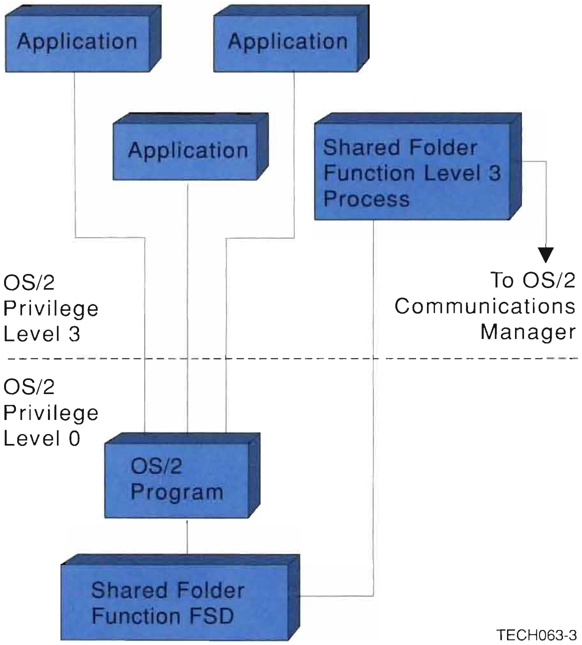


Figure 31. *OS/2 Shared Folders Flow*

The shared folder function FSD is an OS/2 dynamic link library (DLL). This program is called EHNSFL0.DLL. The OS/2 operating system calls EHNSFL0.DLL to perform a request for a shared folder function drive. The OS/2 operating system is a multitasking operating system, but only one request for a drive can be handled at a time. EHNSFL0.DLL queues the request until it can be processed. The OS/2 operating system is also a protected operating system (the resources of one program are private to that program). EHNSFL0.DLL runs at privilege level 0 and has access to the application's data areas. While it has access to these data areas, it makes a copy of all data and necessary control information for a request. It gives this data to the part of the shared folder function that communicates with the AS/400 system when the request is processed.

The part of the shared folder function that communicates with the AS/400 system is the EHNSFL3.EXE program. This program is an OS/2 application that runs as a background process. (An OS/2 process is a running program plus the resources it uses.) This program receives a request from EHNSFL0.DLL, formats the request into the proper DDM data stream, and sends the data to the AS/400 system by calling the communications manager. EHNSFL3.EXE has two threads per drive. A thread is a running unit contained in a process. Threads can run concurrently and independently, and they share resources of a process. The send thread builds a DDM request and sends it through the communications manager. The receive thread gets the reply from the communications manager and returns the reply to EHNSFL0.DLL. Two threads allow concurrent processing. While the receive thread is waiting for the reply from the AS/400 system, the send thread builds the next request.

The OS/2 DOSFsCtl API ties the two pieces of the shared folder function together. EHNSFL3.EXE makes a DOSFsCtl call to get a unit of work to do. EHNSFL0.DLL receives the call, makes any necessary data available to EHNSFL3.EXE, then returns control to EHNSFL3.EXE. At that point, the DDM request is built and sent to the AS/400 system. When EHNSFL3.EXE receives a reply to the request, it gives the reply back to EHNSFL0.DLL through another DOSFsCtl call.

Network Coexistence

While it is simplest to discuss the AS/400 system and the work station as an integrated pair of systems, in reality, both systems have to be recognized as players in a larger network. The most competitive solutions often combine PC servers with the AS/400 system. A PC server is a personal computer whose resources can be used by other personal computers (or other non-PC requesters). For example, a PC server can allow other personal computers to use its printer or fixed disk. There are functions that a PC server does better than an AS/400 system, just as there are functions that an AS/400 system does better than a PC server.

For Version 2, PC Support has been enhanced to coexist with both Novell Netware and the OS/2 local area network (LAN) server. On both token-ring and Ethernet connections, a personal computer can concurrently:

- Use PC Support to access resources and applications on an AS/400 system

- Use either Netware or DOS LAN requesters to access resources on a PC server (see Figure 32)

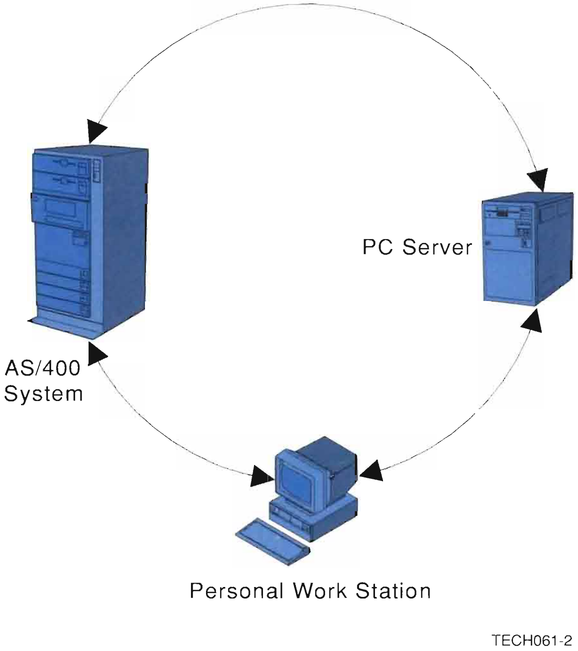


Figure 32. *PC Support Network Coexistence*

Additionally, users can load PC Support programs on an OS/2 LAN server, transparently redirecting requests for either file or printer resources to the AS/400 system.

PC Support Hypertext

In the past, the PC Support help only provided the user with information that was necessary to complete the task on the display. By pressing F1 (Help), the user could get help for the prompt where the cursor was positioned. For more information on the entire display, the user could get general display help by pressing F2 (Extended help).

Now the user can choose hypertext links to get even more information about topics of interest. A **hypertext link** is a method for moving between help modules. These hypertext links are easy to detect when a user looks at a help display. The words that are hypertext links are highlighted in a color different from the rest of the text. Through hypertext links the user can access general information on PC Support, information on a PC Support function, or specific information on a PC Support command.

To use a hypertext link, the user simply moves the cursor to a highlighted word and presses Enter. Another help display immediately replaces the previous display. From this help display, the user may take more links, press Esc to return to the previous help display, or press F3 (Exit) to return to the original display from which F1 was pressed.

Retrievability aids provide users with a means to see where they have been and to avoid moving in a circular fashion. The user can press F6 (List) to see a list of the help displays already viewed using hypertext links. The user can go back to any of those displays by moving the cursor to the display title (a hypertext link) on the list and pressing Enter, or the user can exit from the hypertext search by pressing F3 (Exit).

Hypertext links are flexible in the way information can be chained together. The user can easily move from one type of information to another. For example, the user can move from a syntax diagram to a word definition to a help description.

The hypertext links, in combination with the help information, allow users to determine the amount of detail they want. A user may navigate through a display, fill in the necessary prompts, and never take a hypertext link. However, the more curious user may learn about many PC Support topics. These additional topics describe application integration, communications concepts, and information about many of the PC Support functions.

Conclusion

PC Support/400 provides a wide range of functions to integrate personal computers with the AS/400 system. The enhancements to PC Support since Version 1 Release 1 make the personal computers easier to attach to the AS/400 system and, once attached, the PC Support functions are easier to use. The functions of PC Support also leave more DOS memory for user applications.

Advanced Peer-to-Peer Networking Enhancements

Describes the changes made to the advanced peer-to-peer networking (APPN) support since its initial release on the AS/400 system.

David A. Christenson, Mark A. Cossack, Dennis J. Frett, and John E. McGinn

Introduction

There have been several enhancements made to the APPN implementation on the AS/400 system since its initial release. Refer to the article, "Advanced Peer-to-Peer Networking" in *IBM Application System/400 Technology*, SA21-9540, for background information on the function provided in the initial release of APPN on the AS/400 system. These enhancements provide improvements in usability, availability, connectivity, and performance in many environments. The enhancements discussed in this article include:

- Automatic configuration support of APPC controller descriptions on local area networks (LANs)

  This support allows the AS/400 system to automatically create or vary on controller descriptions when unknown systems connect to the AS/400 system.

- Connection networks

  A connection network takes the automatic configuration support a step further; it allows APPN to dynamically determine addressing information about other systems on the LAN so that configurations for outgoing connections (as well as for incoming connections) can be made.

- Dynamic switching of network node servers

  End nodes have the ability to establish a connection with an alternative network node server without operator intervention or disruption to existing connections.

- Multiple network connectivity

  Multiple network connectivity allows any nodes in adjacent APPN networks (with different network identifiers) to communicate with each other.

- Interactive versus batch line sharing

  Interactive versus batch line sharing enhancements prevent response time degradation of interactive jobs that are sending small amounts of data because of a line being shared with a batch job that is sending large amounts of data.

Automatic Configuration of APPC Controllers on LAN

The AS/400 system now has the capability of automatically creating APPC controller descriptions dynamically when an incoming connection request is received from an unknown system directly connected to the AS/400 system on a LAN. On a LAN protocol, when a system initiates a connection to another system, information about the local system is provided in the initial connection request and the subsequent exchange identification (XID) exchange. This is key to automatic creation of an APPC controller description. The **controller description** is a configuration object that contains information about a remote system (that is, remote system address, APPN control point name, and any other specific parameters required on a link station basis).

On an AS/400 system with automatic creation of APPC controller descriptions on LANs, an APPC controller description is created for any system that initiates a connection to the AS/400 system on the LAN. This is only allowed if automatic configuration has been activated for the LAN line on which the connection request is received.

To visualize the environment suitable for enabling the AS/400 system's automatic configuration support on LAN, refer to Figure 33 on page 67. This figure illustrates the physical configuration of a LAN.

When a remote system initiates a connection request, the AS/400 system first determines if there already is an existing controller description that can be used for the connection. If no controller description currently exists that can be used, the system automatically creates a new object. After the controller description is created, the system automatically varies it on for the connection.

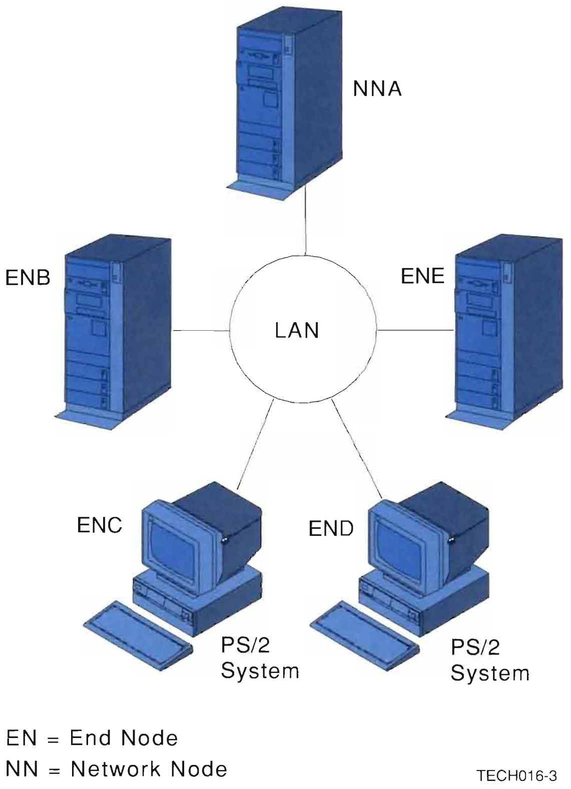


Figure 33. *Example Physical Configuration of a LAN*

Because a varied on controller description is normally needed to supply the required information to the system, connection establishment cannot be delayed while a new controller description is being created. Normal responses must be returned to the remote system to continue with the connection establishment to avoid time-outs by the adjacent system. Supplying this initial configuration information is accomplished by allowing the user to manually create a controller description that is used as a model for all controller descriptions created for a particular resource (LAN line). This model also supplies the values the system needs to bring up the connection. If the user does not supply a model, then system defaults are used.

The user can request that controller descriptions be automatically varied off and deleted after a certain period of inactivity. On an AS/400 system, a user can activate this function by specifying a certain time interval on the automatic delete controller (AUTODLTCTL) parameter of the LAN line description with which these controller descriptions are associated.

Connection Networks

Building on the concept of automatic creation of APPC controller descriptions on LANs is the establishment of direct APPN sessions using dynamic address resolution. This allows for the dynamic determination of configuration information (for both incoming and outgoing connections) so that direct connections can be established between systems on the same shared-access transportation medium (such as a token-ring or Ethernet LAN). To accomplish this, the concept of a connection network is used. A **connection network** (an APPN architectural term defining the use of a shared-access transportation medium for route calculation) attempts to reduce manual configuration and reduce the size of the APPN topology database.

Refer to Figure 34 on page 68 to see the representation of a connection network. The idea behind a connection network is that a shared-access transportation medium is considered a virtual node by all the systems that are part of the connection network. Because the virtual node is considered to be a node in the network, APPN route selection services can calculate routes that go "through" this virtual node. Each node has a transmission group (TG) entry in its topology database that describes its connection to the virtual node. This TG entry contains data link control (DLC) signaling information about itself (that is, for a token-ring network there would be an entry that contains a node's local medium access control (MAC) and service access point (SAP) address).

At the time a route needs to be calculated, APPN route selection services determines that two nodes requesting a route each have connections to the same virtual node. The routing information that is returned to the originator of the route request contains the DLC signaling information of the destination. The originator can then establish a direct outgoing connection to the destination (because the originator now has the address of the destination). In order for the node to provide this information to the network node, it must have an APPN control-point-to-control-point (CP-CP) session established with the network node. This is a requirement for a node to participate in a connection network.

It is also important to mention that a connection network may also be used for intermediate routing purposes between two APPN network nodes that do not have prior configuration information about each other.

The following is a discussion that emphasizes the key concepts in connection networks.

Figure 33 on page 67 shows an environment suitable for the use of a connection network. Assume there are five systems on the token ring. Prior to having connection network support, if any system on the ring wanted to have direct communications with any other system on the ring (that is, without having the APPN network node, NNA, performing intermediate routing), each system would be required to have one token-ring line and four configuration objects describing each of the remote systems. As more systems are added to the network, this becomes a configuration burden.

Refer to Figure 34 for a logical view of a LAN using a connection network. In this case, each system only needs one token-ring line and one APPC controller description manually configured. The APPC controller description needed is called a model controller description. A model controller is indicated by specifying the model controller (MDLCTL) parameter on the Create APPC Controller (CRTCTLAPPC) command. In this object, the line, the name of the virtual node, the control point name, and the network identifier of the system that a CP-CP session needs to be established with (that is, the end nodes would specify NNA) is configured.

Assume ENB wants to establish a logical unit (LU 6.2) session with ENE. ENB requests a route to ENE from its network node server (NNA). ENB reports to NNA that it has a transmission group (TG) to VN1, and the DLC signaling information associated with its connection is 2. NNA determines that the location name being searched for resides on ENE (using base APPN search logic), so it sends a search request to ENE. On the search reply, ENE reports its TG information (one of which is the TG from ENE to VN1, which has DLC signaling information of 5). During the route calculation phase, NNA determines that both ENB and ENE have TGs to the same connection network. Assuming that the connection through the virtual node is the most optimal route based on class of service, NNA returns a route to ENB indicating that the optimal route is ENB to VN1 to ENE. Because the route traverses a virtual node, the DLC signaling information from VN1 to ENE is supplied. ENB can now automatically create an APPC controller description and make the outgoing connection to ENE. ENE can accept the incoming call and automatically create its controller description. Because these connections are brought up through a virtual node, ENB and ENE do not include the direct link just established into their topology databases. The TG representing the direct link is not required for future session initiation requests because the route through the virtual node can be used at that time instead.

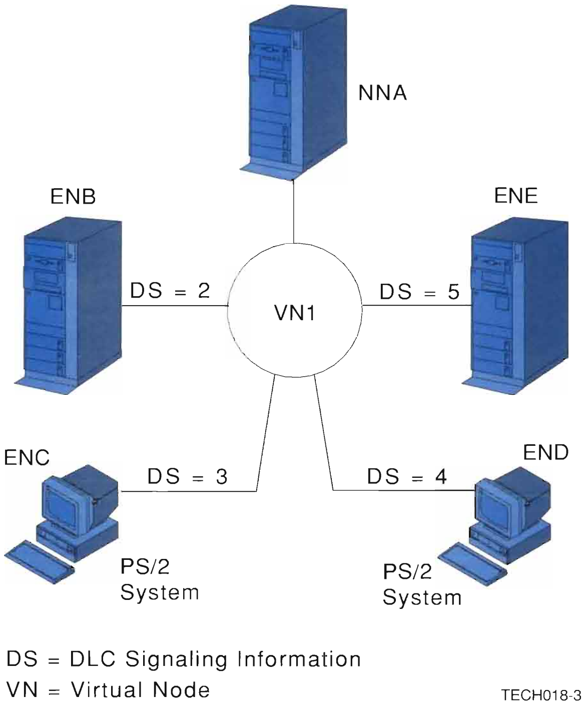


Figure 34. *Logical Configuration of a LAN Using a Connection Network*

As the example illustrates, ENB and ENE establish direct connections without having any configuration information about each other. The number of node and TG entries in the topology database of a system that supports connection networks and the number of topology updates that are broadcast throughout the network is significantly reduced with connection networks.

Dynamic Switching of Network Node Servers

An APPN end node requires a CP-CP session established with a network node server to participate in directory searches, to establish APPC sessions that use the most optimal routes, and to send alert data to its network node server.

When an end node loses its CP-CP session with its network node server, the end node is logically disconnected from the network. The loss of a CP-CP session between an end node and its network node server can be caused by communications line failures, varying off of configuration objects, or a hardware failure on the network node server.

The only way for the end node to logically reconnect to the network is by reestablishing its CP-CP session with the original network node server or with an alternative network node server that is directly attached. Prior to Version 2 of the AS/400 system, an operator at an end node was required to force a link level exchange (XID exchange) between itself and a network node server to reestablish its CP-CP session. All active user sessions on the link needed to end for this XID exchange to occur. This approach for changing network node servers is undesirable because it requires operator intervention and it is disruptive if user sessions are active on the affected connection.

Functional Description

The enhancement of dynamically switching network node servers is provided in Version 2 of the AS/400 system. This enhancement provides CP-CP session recovery with an alternative network node server following a CP-CP session outage with the original network node server. This CP-CP session recovery is provided without operator intervention, and it eliminates the disruption of active user sessions to carry out the task. In the past, establishment of CP-CP sessions was tied directly to the XID exchange. The primary change required to support end node dynamic switching of network node servers is to remove the dependency of these XID exchanges for CP-CP session establishment.

Nodes that support dynamic switching of network node servers are considered **up-level nodes**. Nodes that do not support dynamic switching of network node servers are considered **down-level nodes**. An up-level end node indicates during its XID exchanges that it is requesting services from every adjacent node (assuming the adjacent node is in the end node's network node server list). The first network node in the server list that the end node connects to and that offers services to the end node is the system that the end node establishes its CP-CP session with. As the end node completes XID exchanges with other network nodes, it determines that it already has a server established, and the end node does not initiate CP-CP sessions with the other network nodes.

A network node that supports this function does not automatically establish its CP-CP session following an XID exchange with an end node. The network node waits until the end node starts its CP-CP session with the network node before the network node starts its CP-CP session to the end node. The end node is in control of which system it establishes its CP-CP session with and when it establishes it.

When an end node loses its CP-CP session with a network node, the end node attempts to reestablish the CP-CP session with the same network node (if a parallel TG exists to that network node). If this is unsuccessful, the end node chooses the first network node in the network node server list that meets the following criteria:

- The end node has an active TG to the network node.
- The network node is up level.

Example of Switching Network Node Servers

To illustrate the functional description, refer to Figure 35 on page 70. Assume that Portland, Houston, and Seattle are all up-level systems, and that Chicago is down level. Also assume that Houston, Seattle, and Chicago are listed in Portland's network node server list. The following steps describe bringing up the APPN network from Portland's perspective, and describes the outage of a CP-CP session.

1. Portland establishes a connection with Seattle. Because Seattle is in the server list and Portland has no server, Portland establishes its CP-CP session with Seattle. After Portland establishes its CP-CP session, Seattle then establishes its CP-CP session with Portland.
2. Portland establishes a connection with Houston. Because Portland already has a server established, it does not establish a CP-CP session with Houston at this time. Since Houston is an up-level system, it does not attempt to establish its CP-CP session with Portland.
3. Portland establishes a connection with Chicago. Since Portland already has a server established, it does not establish a CP-CP session with Chicago at this time.

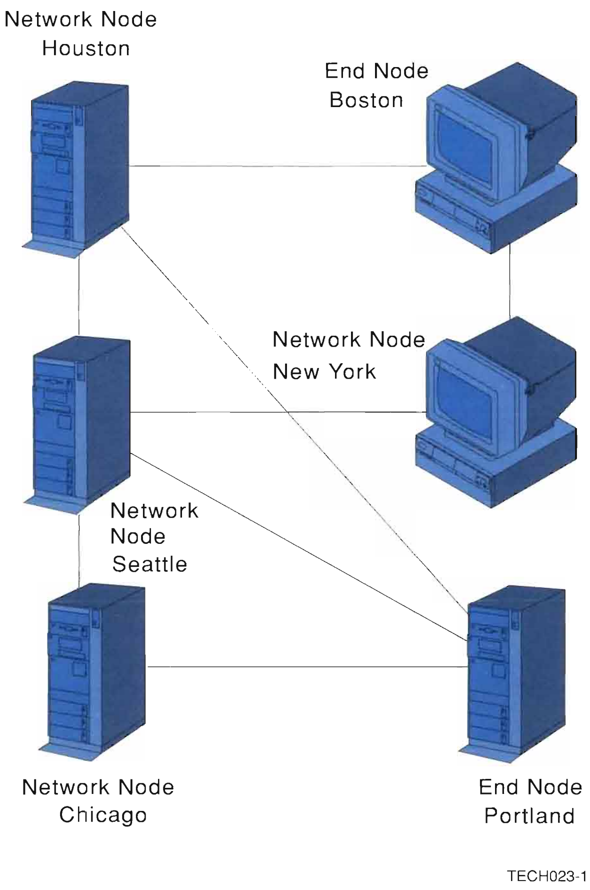


Figure 35. *APPN End Node with Multiple Network Node Servers*

Because Chicago is a down-level system, it attempts to establish its CP-CP session with Portland. Upon receiving this request from Chicago, Portland rejects the request with a sense code indicating the end node is already being served.

4. A link outage occurs between Portland and Seattle. The link outage causes the CP-CP sessions to be lost.

5. Because Portland does not have an alternative link to Seattle, Portland searches through its network node server list to find a system that matches the search criteria. Chicago is a down-level system; therefore, the only eligible choice is Houston. Since the APPC controller description on Portland is already active for Houston, Portland establishes its half of the logical CP-CP session. Upon receiving this request, Houston then establishes its CP-CP session with Portland. At this point, Portland is once again in contact with the APPN network.

APPN Multiple Network Connectivity

Different APPN networks are represented by having each network maintain a unique 8-character network identifier (NETID). These different APPN networks may represent separate enterprises. There are times when APPC applications in one network need to access logical units that reside in a different APPN network. This could be common if one enterprise takes over another, or if a service provider, such as a third-party vendor, needs to provide services to many different enterprises (each with different network identifiers).

Before APPN Multiple Network Connectivity

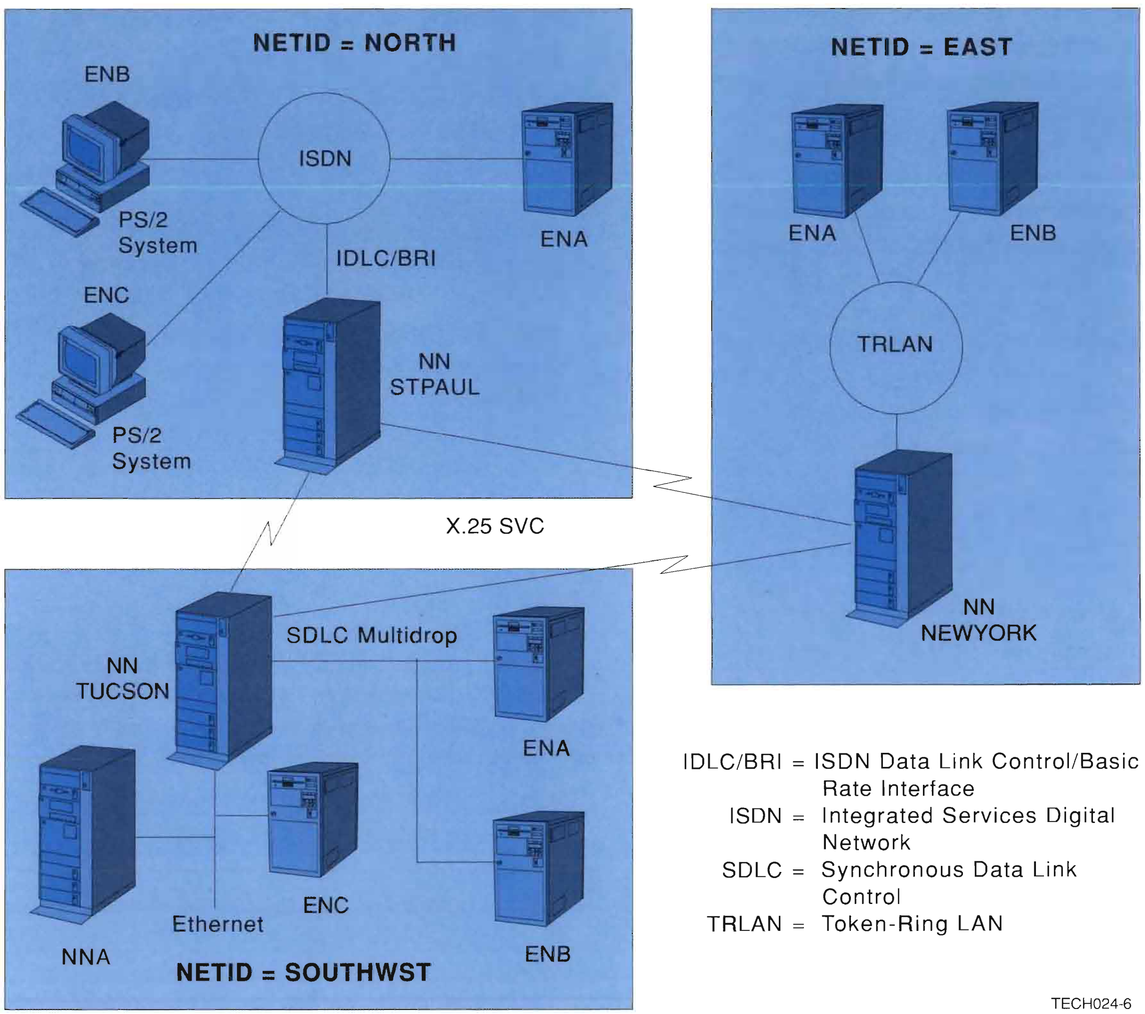
Prior to APPN multiple network connectivity, there was limited connectivity between APPN networks with different NETIDs. This limitation was imposed by not allowing two APPN network nodes with different NETIDs to establish a link-level connection and subsequently a CP session (which carries APPN control traffic). This limitation was necessary because there was no method that allowed these networks to interconnect and allow complete access to all logical units without sharing the complete topology of both networks.

After APPN Multiple Network Connectivity

With Version 2 Release 1 of the AS/400 system, APPN network nodes have been enhanced to support full connectivity between APPN networks that are directly connected (adjacent) to each other and that have different NETIDs.

The connectivity without topology overhead is accomplished by having the Version 2 AS/400 network node portray the image of an end node to the network nodes it is attached to in adjacent APPN networks (nonnative networks). The network node still functions as a normal network node in its native network. Because a Version 2 AS/400 network node portrays itself as an end node to the nonnative networks, the topology isolation requirement is automatically taken care of. Topology exchanges do not occur between an end node and a network node.

Figure 36 on page 71 illustrates some of the connectivity options available with APPN. Notice that there are three APPN networks each with a unique NETID. There is complete connectivity between all logical units (LUs) on every system in the diagram without the need to manually define any nonnative LUs, and at



IDLC/BRI = ISDN Data Link Control/Basic Rate Interface
ISDN = Integrated Services Digital Network
SDLC = Synchronous Data Link Control
TRLAN = Token-Ring LAN

Figure 36. *Multiple APPN NETID Networks Fully Interconnected*

the same time the topology of each network stays independent. In other words, there is no exchange of topology between the NORTH NETID network and the EAST NETID network, and yet LU 6.2 (APPC) sessions can be established from NORTH.ENA to EAST.ENB while devices are automatically created for the new session.

All of the links interconnecting the three APPN NETID networks are needed because Version 2 APPN multiple network connectivity supports connectivity between adjacent networks only.

APPN Multiple Network Connectivity Logic

The additional functionality added by APPN multiple network connectivity involves changes to link establishment, directory search, and bind processing.

Link Establishment: During link establishment, if a network node determines the system it is attaching to is in its own (native) network, then it tells the adjacent system it is a network node during its exchange identification flows (XID). If a network node attaches to a system in a nonnative network, it:

- Negotiates the role of end node or network node if the other system is also an AS/400 network node at Version 2
- Assumes the end node role if the other system does not support APPN multiple network connectivity
- Assumes the network node role if the other system is an end node

The general idea is that when a network node connects to a node in a nonnative APPN network, one system takes the end node role and the other system takes the network node role.

Directory Search Processing: Before describing changes to search processing for APPN multiple network connectivity, it is necessary to describe what role an end node performs depending on whether it is the source or target of a session initiation request. When an end node is the source of a request, it forwards the request to its network node server, supplies links that it has to network nodes, and expects the network node server to search for the owning control point of the location and return to the end node a complete end-to-end route. When an end node is the target of a session initiation request, the end node receives a request from its network node server to verify that a location resides on the end node, and the end node also supplies link information so that the network node server of the source node can calculate a complete end-to-end route.

When searches originate in the native network and are destined for a nonnative network, the Version 2 AS/400 system that is connected to a network node in the destination nonnative network transforms the search to appear as though it were being sent by an end node (that is, it supplies link information, among other things). When the Version 2 AS/400 system receives a search response from the nonnative network node, it stores the routing information and responds to the search in the native network as though the location were found in an attached end node (that is, the entire nonnative network is portrayed as an end node to the rest of the nodes in the native network).

When the search originates in the nonnative network and is received by the native Version 2 AS/400 system, the system acts like the network node server of an attached end node and attempts to find the location in the native network. If the location is found, the native Version 2 AS/400 system calculates a route from itself to the target, and stores the routing information for later use in BIND processing. When sending the search response back to the nonnative network node, the native Version 2 AS/400 system indicates that the location was found and modifies the search response as though it were being returned by an end node. The nonnative network node forwards its response to the source node, which causes a route to be calculated from the source node in the nonnative network to the native Version 2 AS/400 system.

Bind Processing: Once search processing is complete, the bind request needs to be sent from the source system to the target system. As described earlier, the complete route actually consists of two separate routes (one for routing through the native network and one for routing through the nonnative network). When the Version 2 AS/400 system receives a bind request from a nonnative network node or the received bind is destined for a nonnative network node, the local network node removes the routing information it receives, and replaces it with the stored routing information. In the event that the stored routing information is no longer stored when the bind request is received, the Version 2 AS/400 system takes the necessary steps to forward the bind to the destination.

Interactive versus Batch Line Sharing

Integration of interactive and batch (mixed) traffic on the same communications line is often necessary. For example, a user at an interactive work station may concurrently share the same line with a job that is transmitting a large file. Interactive traffic typically consists of small, sporadic two-way transmissions, versus the large continuous one-way transmissions of batch traffic. An interactive response time ranging from 0 to 3 seconds is preferable, but it cannot be achieved if batch traffic is allowed to consume too much of the line bandwidth. Some methods to effectively integrate mixed traffic are discussed, but first some background information is provided.

It is important to understand the flow of session traffic through the transport layer of the Systems Network Architecture (SNA). The transport layer consists of two vertical components. The first component is either a **transmission control** (end point session) or **session connector** (intermediate session), and its major function is session-level pacing. Some of the transmission controls and session connectors may transmit interactive traffic and others batch traffic. The second component is **path control**, and one of its major functions is to prioritize the transmission order when multiple transmission controls and session connectors share the same path control. Session traffic is packaged into **request units** as it enters the transport layer and then into **frames** prior to passing through **data link control** (which manages the transmission of data over the physical communications line). As session traffic passes through the transport layer, its transmission may be temporarily blocked while it is waiting in a pacing queue or transmission priority queue. Increasing the queuing time of batch traffic relative to interactive traffic is the key to effective integration of mixed traffic. Figure 37 shows the flow of session traffic through the transport layer.

SNA Session-Level Pacing

SNA session-level pacing is a flow control technique that permits a receiving session to control the data transfer rate (the rate at which it receives request units) on the normal flow. Its primary use is to prevent overloading a receiver with unprocessed requests when the sender can generate requests faster than the receiver can process them. If fixed pacing is used, the window size is fixed for the duration of the session. APPN supports adaptive pacing, which allows a variable window size.

Fixed pacing can be used to slow the flow of batch traffic and provide good interactive response time. Batch traffic is assigned a small window size of 1 or 2 and interactive traffic is assigned a larger window size of 7 or more. Fixed pacing has the following disadvantages:

- The flow of batch traffic is restricted even when interactive traffic is not present, reducing throughput.
- Multiple batch sessions can collectively consume the bandwidth, causing poor interactive response time. For example, 7 batch sessions with a window size of 1

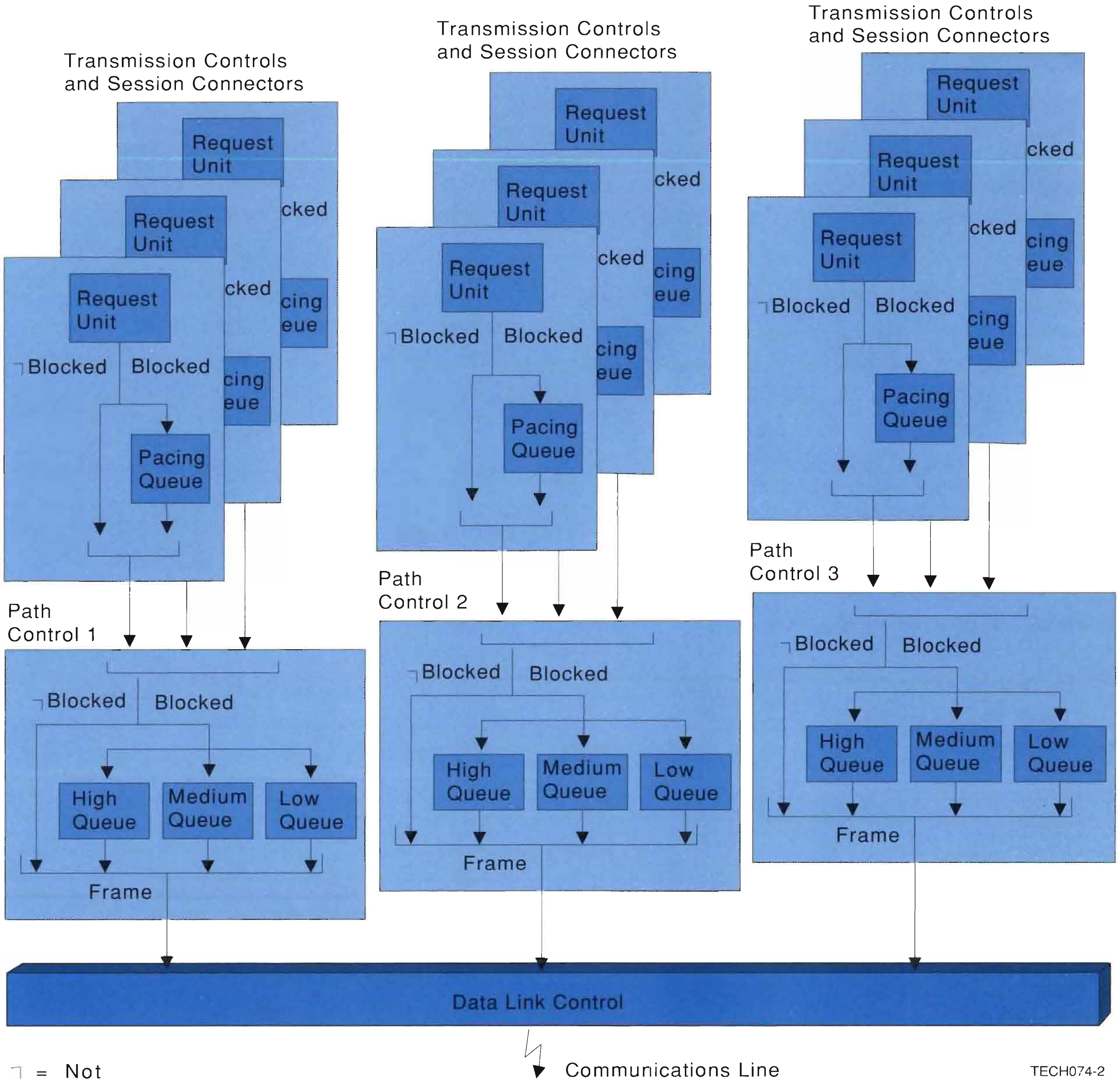


Figure 37. *Transmission of a Request Unit*

would allow up to 7 request units to be concurrently in transit.

Adaptive pacing provides a more dynamic and responsive means of controlling congestion problems in a large network. The goal of adaptive pacing is to control congestion and fairly allocate available buffer resources to each session. As long as there is minimal congestion, a session is granted a pacing window based on the supply and demand of buffers. Because the demand for buffers by the interactive sessions is low, the batch sessions are granted large window sizes and are able to easily consume the bandwidth of a slow-speed line, degrading interactive response time. Therefore, adaptive pacing does not provide a mechanism to effectively integrate mixed traffic.

Internal Session-Level Pacing

The APPN support now provides internal session-level pacing to pace the flow of request units sent to path control for adaptively paced sessions. It is not part of SNA because it does not cause any external line flows. The sender is allowed to send a limited number of request units to path control and is not allowed to send any more until a request unit is successfully delivered to the adjacent system.

Batch traffic is assigned a small request unit limit of 1 or 2, and interactive traffic is assigned a larger request unit limit of 7 or more. The request unit limit is configured in the INPACING and OUTPACING parameters of the mode description.

Internal pacing regulates mixed traffic in a manner similar to SNA fixed pacing, but it does not have the overhead of additional external line flows (for example, fixed pacing sends a pacing response). It does have the same disadvantages as stated previously for SNA fixed pacing.

Transmission Priority

Transmission priority allows different priority levels to be assigned to session traffic that determines the precedence for being selected by path control for transmission to the adjacent system. Path control provides three user-defined priority levels: high, medium, and low. Interactive traffic is typically assigned high priority and batch traffic, medium or low priority.

The transmission priority algorithm used since Version 1 determines the servicing order based on priority level and arrival order. Each priority level is assigned a different priority number:

High = 0
Medium = 50
Low = 200

An arrival number (one greater than the previous) is assigned to each request unit as it arrives in path control, and then it is enqueued to the proper priority queue in first-in first-out order. The selection of the next request unit sent to data link control is determined as follows:

1. A service number is calculated for the first request unit in each priority queue:
   service number = priority number + arrival number
2. The request unit with the smallest service number is selected.

Prior to Version 2, transmission priority was ineffective in a mixed traffic environment. In Version 2, an enhancement provides sensitivity to the response time of high priority frames.

Frame response time is a measurement of the time it takes to send a frame to the adjacent system. The time interval starts when a frame is sent by path control to data link control and ends when path control is notified that the frame has been successfully delivered to the adjacent system. Frame response time is categorized into the following ranges:

Very poor	Greater than 2 seconds
Poor	Between 1 and 2 seconds
Acceptable	Between 1/2 and 1 second, inclusive
Good	Less than 1/2 second

The goal is to keep the frame response time in the acceptable range. This is achieved by limiting the maximum number of **outstanding frames** (a frame that has been sent to data link control and has not yet been delivered to the adjacent system) of medium and low priority. The **medium/low maxout** (the maximum number of outstanding frames of medium and low priority) is dynamically adjusted based on the frame response time of each high priority frame.

The scope of the enhanced transmission priority algorithm spans all path controls sharing the same data link control. Separate priority queues are still maintained by each path control, but the same medium/low maxout is adjusted by each path control. For example, if a path control transmitting only interactive session traffic detects poor response time, it reduces the medium/low maxout to slow the flow of batch session traffic for the other path controls sharing the same data link control. Therefore, the medium/low maxout is collectively determined by all path controls sharing the same data link control.

The maximum number of outstanding frames associated with a single data link control cannot exceed P * M, where:

P = the number of path controls sharing data link control
M = the medium/low maxout

Figure 38 shows how the medium/low maxout is adjusted by path control due to changes in the response time of high priority frames. Figure 39 on page 76 shows how the unrestricted flow of medium and low priority frames is restored by path control after more than 5 minutes without any high priority frame transmissions. A detailed description of the flowchart steps in both figures follows.

Step 1. Data link control notifies path control when an outstanding high priority frame is successfully delivered to the adjacent system.

Step 2. Poor response time is quickly remedied by limiting the number of outstanding medium and low priority frames to one. If the frame response time improves, the medium/low maxout is allowed to gradually increase again.

Step 3. Because poor response time is just above the acceptable range, only a slight decrease in the medium/low maxout is necessary to push frame response time back into the acceptable range. If the medium/low maxout (maximum number of outstanding frames of all priorities) is currently greater than or equal to the **overall maxout**, the medium/low maxout is set to one-half of the overall maxout. Otherwise, the medium/low maxout is only decreased by one.

Step 4. Acceptable response time is within the targeted range; therefore, the medium/low maxout is not changed.

Step 5. Good response time may mean that medium and low priority transmissions are being unnecessarily limited; therefore, the medium/low maxout is gradually increased by one.

Step 6. The time of delivery of the most recent high priority frame is saved and used in step 8.

Step 7. Data link control notifies path control when an outstanding medium or low priority frame is successfully delivered to the adjacent system.

Step 8. The difference between the current time and the time of delivery of the most recent high priority frame (step 6) is calculated to give the high priority inactivity time used in step 9.

Step 9. If the high priority inactivity time is less than or equal to 5 minutes, the value of the medium/low maxout is not changed.

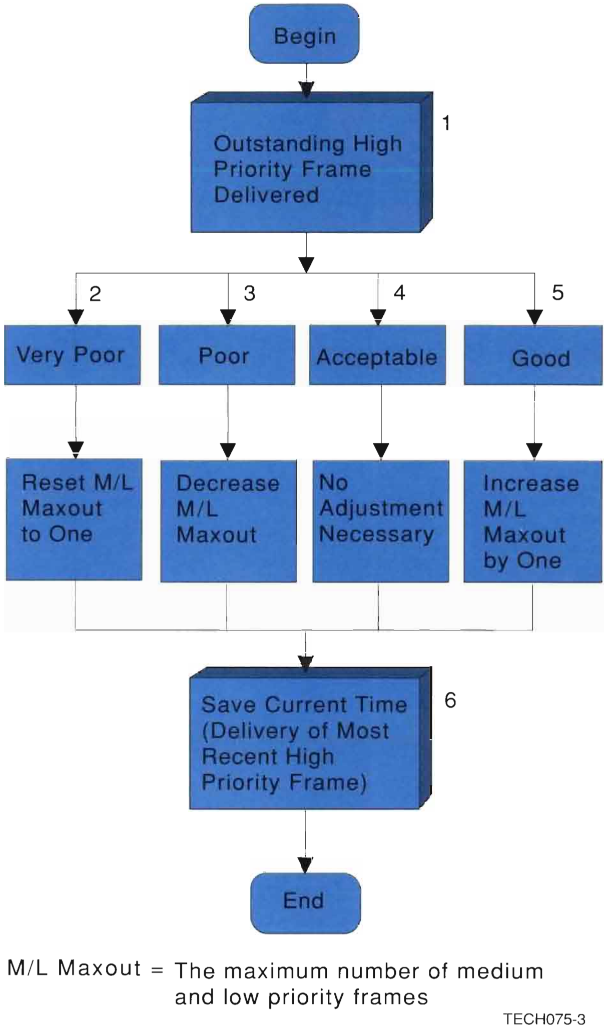

Begin
1
Outstanding High Priority Frame Delivered
2
3
4
5
Very Poor
Poor
Acceptable
Good
Reset M/L Maxout to One
Decrease M/L Maxout
No Adjustment Necessary
Increase M/L Maxout by One
6
Save Current Time (Delivery of Most Recent High Priority Frame)
End
M/L Maxout = The maximum number of medium and low priority frames
TECH075-3

Figure 38. *Restricting Flow of Medium and Low Priority Frames*

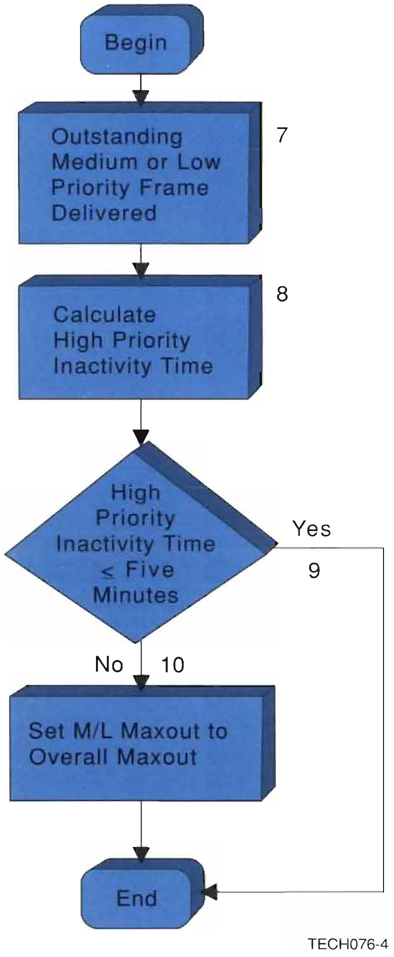

Begin
Outstanding Medium or Low Priority Frame Delivered
7
Calculate High Priority Inactivity Time
8
High Priority Inactivity Time ≤ Five Minutes
Yes
9
No
10
Set M/L Maxout to Overall Maxout
End
TECH076-4

Figure 39. *Restoring Unrestricted Flow of Medium and Low Priority Frames*

Step 10. If the high priority inactivity time is greater than 5 minutes, the medium/low maxout is set to 65535 to allow unrestricted transmission of medium and low priority frames. Medium and low priority frames can now be freely transmitted until the response time of a high priority frame transmission falls below the acceptable range.

Transmission priority has the following advantages over fixed session-level pacing and internal session-level pacing:

- The flow of batch traffic is not restricted after interactive traffic is inactive for more than 5 minutes.
- In most environments, multiple batch sessions are not allowed to collectively consume the bandwidth.
- The flow of batch traffic is dynamically adjusted, allowing batch traffic to use as much of the bandwidth as possible without causing poor interactive response time.

Request Unit Size

The request unit size selected for batch session traffic has an effect on interactive response time. If fixed session-level pacing, internal session-level pacing, or transmission priority does not provide an acceptable interactive response time, the request unit size for the batch sessions may need to be reduced. All three methods limit the number of blocks (request units or frames) that can be transmitted by batch sessions. The total amount of data allowed to be transmitted by batch sessions is equal to N * B, where:

N = the number of blocks allowed
B = the block size

Decreasing the number of blocks or the block size lowers the bandwidth consumption. Therefore, decreasing the request unit size (block size) of the batch sessions can improve interactive response time. However, a smaller request unit size can cause additional processing overhead and reduced throughput.

Conclusion

Advanced peer-to-peer networking support has had several enhancements since its initial release. These enhancements provide improvements in some of the key areas of the operating system. These key areas are usability, availability, connectivity, and performance.

Automatic configuration support of APPC controllers on local area networks and connection networks provides usability enhancements because a significant amount of manual configuration is eliminated. The end node dynamic switching of network node servers provides an availability enhancement. This enhancement provides automatic recovery of control point session outages so that end nodes do not lose access to the APPN network.

Multiple network connectivity is an enhancement that expands the connectivity of the AS/400 system in APPN networks. This allows systems in one network to access systems in other APPN networks with little or no configuration changes. The interactive versus batch line sharing enhancements provide a performance improvement by not allowing applications that send large volumes of data to take control of the communications link. This interactive versus batch line sharing allows interactive jobs to keep acceptable response times.

The AS/400 continues to improve its peer-to-peer networking support. With the release of the APPN architecture and the announcement of APPN support on the Personal System/2* system and the 3174 Subsystem Control Unit, the AS/400 system can provide networking support for many different customer environments.

Open Systems Interconnection

Describes the open systems interconnection (OSI) reference model and the OSI Communications Subsystem/400 licensed program, which provides OSI support on the AS/400 system.

Jeffrey C. Kramer, Charles C. Shih, and Kenneth A. Cook

Introduction

During the past 10 years, technological advances have dramatically reduced the cost of computing power, related display stations, peripherals, and automated industrial equipment. This has contributed to the explosive growth of the number of display stations, computers, and related equipment in use. At the same time, many new vendors have entered the marketplace, often with new technologies that are not compatible with existing equipment.

Over the years, manufacturers have responded to their customers' needs to communicate among departments, offices, branches, corporate divisions, cities, countries, and, indeed, continents. For many computer and display station vendors, this effort started in the early 1970s, years before multiple-vendor communications standardization had been considered. Consequently, vendors developed their own proprietary network architectures and sets of system protocols. These proprietary networks were tuned and optimized for specific operating environments.

Through various information processing requirements and acquisition practices, many organizations found themselves with computer equipment from different vendors, equipment that could not communicate. As the number of different display stations, computers, and networks increases worldwide, the need for multiple-vendor, international communications standards also increases. International communications standards are a key factor in the advancement of the information technology industry.

OSI Standards

Central to the standardization effort is the International Organization for Standardization (ISO). The data communications standards evolving from this organization are known as the open systems interconnection (OSI) standards.

In 1977, ISO Technical Committee 97 started to define a framework and structure for a set of rules, called protocols, that would enable systems supplied by different vendors to connect and communicate. A **protocol** is a set of rules that govern data communications; for example, a protocol might allow only one program at a time to send data.

Open systems interconnection (OSI) defines standards by which computers that are basically incompatible can work together. OSI describes the architecture, protocols, and services needed to achieve this goal. Many of these standards are complete, while others are evolving.

In OSI, an **open system** is a computer system that can communicate with another computer system using these OSI protocols. Open systems interconnection is based on a layered architecture called the OSI reference model. Each layer in the OSI reference model has a name, a number, protocols that provide specific functions, and defined services for each protocol. Because the intended range and scope of OSI is broad—from display stations and personal computers to large mainframes—numerous options are available at each layer.

The OSI Reference Model

The OSI reference model consists of seven layers as shown in Figure 40 on page 79. Each layer is responsible for a specific set of functions.

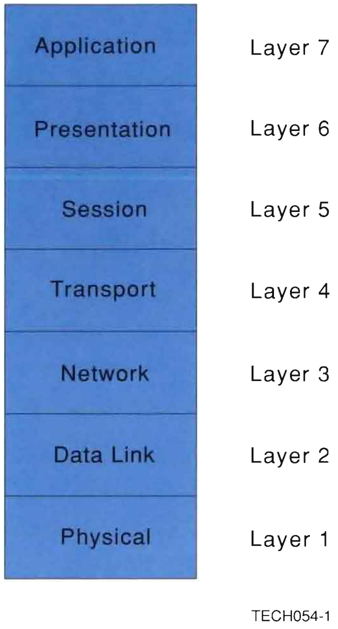


Figure 40. *The OSI Reference Model*

Application Layer

The application layer manages communications between end users on connected open systems and is the OSI layer that directly supports the end user. There are multiple application layer standards, each oriented to its own application processing requirements. Some layer standards have already been defined, while others are still evolving. Among the standards in place are those for:

- File transfer, access, and management (FTAM)
- Message handling systems (X.400)

Presentation Layer

The presentation layer provides services so applications that communicate over different open systems can correctly interpret the data exchanged. This is necessary because different vendors' systems use various methods for internal representation of data, and because different programming languages may be used.

Session Layer

The session layer coordinates the dialogue between users on open systems. This includes functions for negotiating, establishing, and releasing sessions between users.

Transport Layer

The transport layer provides the end-to-end control of data exchanged between users on open systems. It provides the required level of reliability, based on user needs and network- and transmission-error-recovery capabilities.

Network Layer

The network layer provides the addressing, routing, and relaying of information required to provide the data flow between open systems. Data flow may involve establishing, maintaining, and ending connections between open systems. This layer provides services that keep the transport layer independent of the data transmission technology and provides for both connection-oriented (connection setup required) and connectionless (no connection setup required) network protocols.

Data Link Layer

The data link layer provides services for the error control and synchronization of data as it is transmitted over the physical medium between adjacent nodes in the network.

Physical Layer

The physical layer describes the electromechanical characteristics for attachment of open systems to the physical medium. These attachment standards include plugs, sockets, and the encoding of data into suitable electronic signals.

Management and Directory Service Standards

OSI management standards provide for both onsite (local) and centralized (remote) network management. The standards define the management services required to report events and issue commands, and define the protocols to use when the service user is on a remote node. Management information is sent from multiple remote open systems to a centralized management node.

OSI directory standards define the protocols to set up and maintain directory information in an OSI directory. An **OSI directory** contains information to locate entities—application programs, for example—within an OSI network.

The management and directory standards are considered to reside within the application layer.

Peer-to-Peer Communications

The OSI reference model describes a **peer-to-peer communications environment** in which a layer entity of one open system communicates through a set of protocols with its peer entity in the corresponding open system. A **layer entity** is the representation of one of the seven layers within a given open system. This is illustrated in Figure 41.

Each layer within an open system receives services from the layer directly below it and provides services to the layer directly above it. The layer providing the services is called the **service provider**; the layer using the services is called the **service user**.

For example, assume the session layer entity at Open System A wants to communicate with its peer entity, the session layer entity at Open System B. (See Figure 41 for an example of both systems.) To do so, Open System A uses the services of its transport layer entity, its service provider. The transport layer entity, in turn, uses the services provided by its network layer entity. This process continues down through all the Open System A layers. At the lowest level, the physical layer entity at Open System A sends the data to its peer, the physical layer at Open System B. The physical layer entity receives the data on Open System B, then passes it up to the data link layer entity. This continues up through the layers until the session layer at Open System B receives the data.

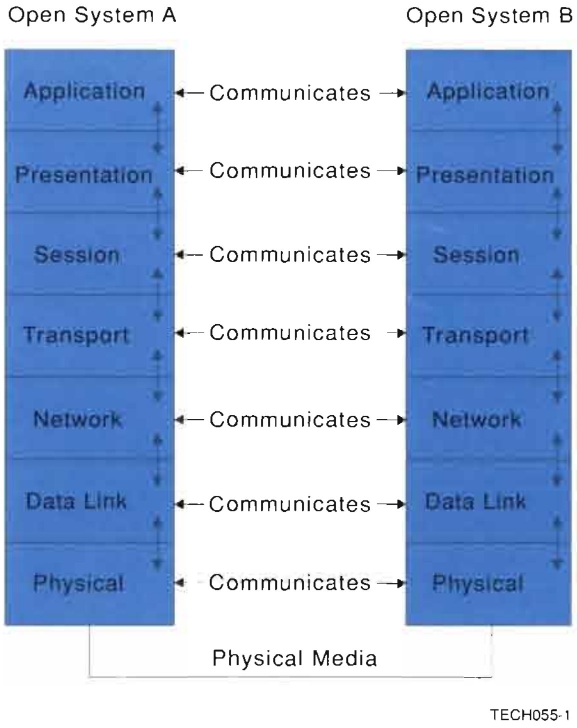


Figure 41. *Peer-to-Peer Communications*

The peer layers communicate *physically* through physical communications media, such as a communications line, but they communicate *logically* peer to peer. Thus, the session layer entity at Open System A logically communicates with, and only with, the session layer entity at Open System B.

Because each layer communicates with its peer layer, it is important that both open systems implement a compatible set of protocols and options at each layer.

OSI Communications Subsystem/400

The OSI Communications Subsystem/400 licensed program enables applications to communicate across multiple vendor systems by providing an open system that conforms to the OSI standards for communications. The licensed program provides a base for the most commonly required OSI applications, such as message transfer, file transfer, and customer-written OSI applications.

OSI Communications Subsystem/400 is part of a family of OSI products that provide consistent OSI support across IBM Systems Application Architecture (SAA) platforms by providing a single OSI protocol implementation across those systems. This helps customers by providing a consistent interpretation of evolving OSI standards.

The structure for the OSI Communications Subsystem/400 licensed program is shown in Figure 42 on page 81.

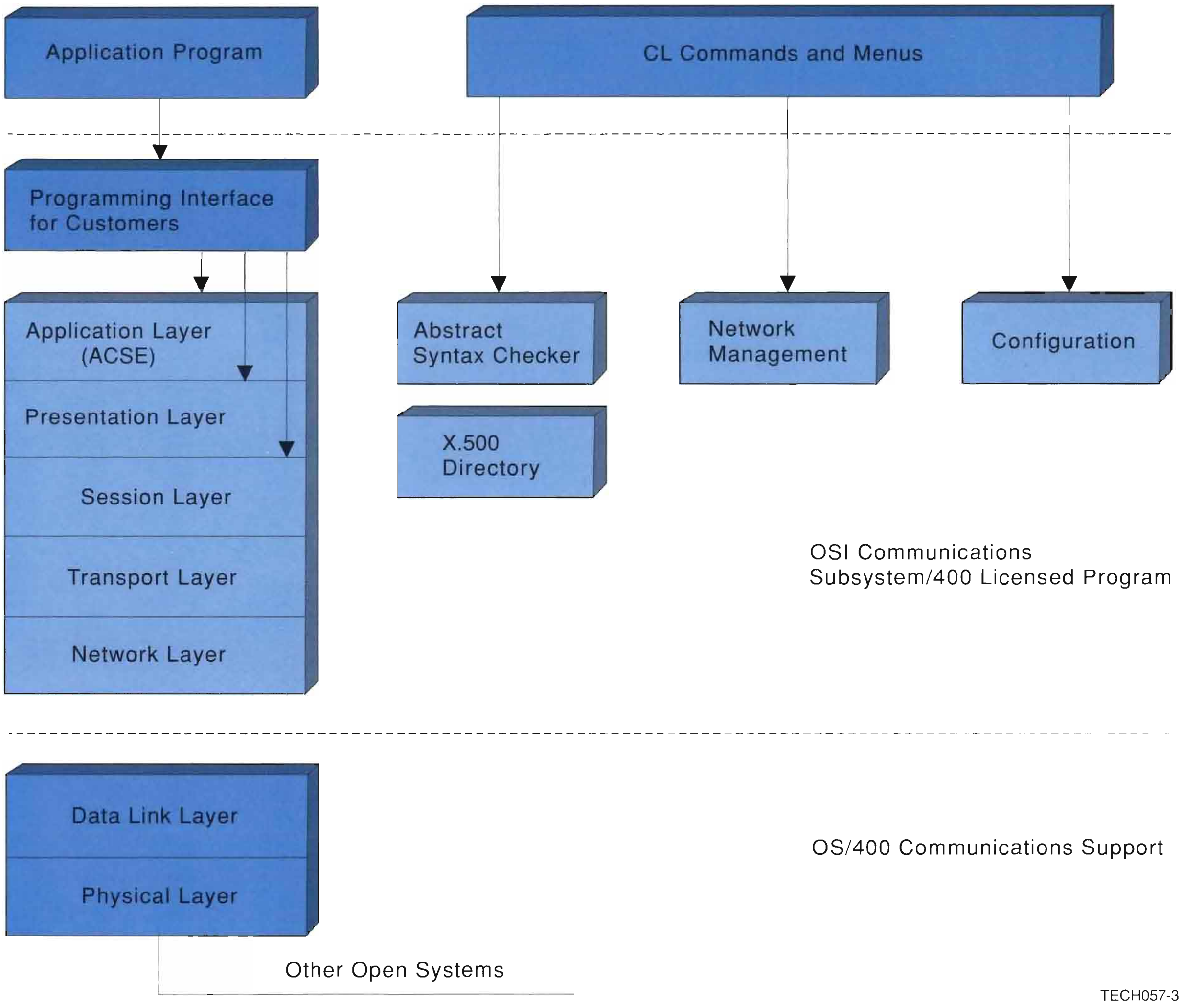


Figure 42. *OSI Communications Subsystem/400 Licensed Program Structure*

Programming Interface for Customers

Information obtained through the OSI layers must be processible by customer application programs. This processing is either through OSI-defined application layer (layer 7) programs (for example, FTAM, X.400, and other interfaces) or user-written application programs. A high-level programming-language interface is provided that can be used with either COBOL or C programs. Through this interface, customers' programs can access services provided by:

- Association control service element (ACSE)
- Presentation layer
- Session layer

The programming interface consists of a set of **callable services**, procedures that can be called from application programs, and is consistent across the SAA environments. The interface allows implementation of simple communications programs with relatively little system-specific programmer education while still providing full-function capability. Through the use of a predefined set of processing options, most procedures have only a few parameters that must be specified.

Abstract Syntax Checker

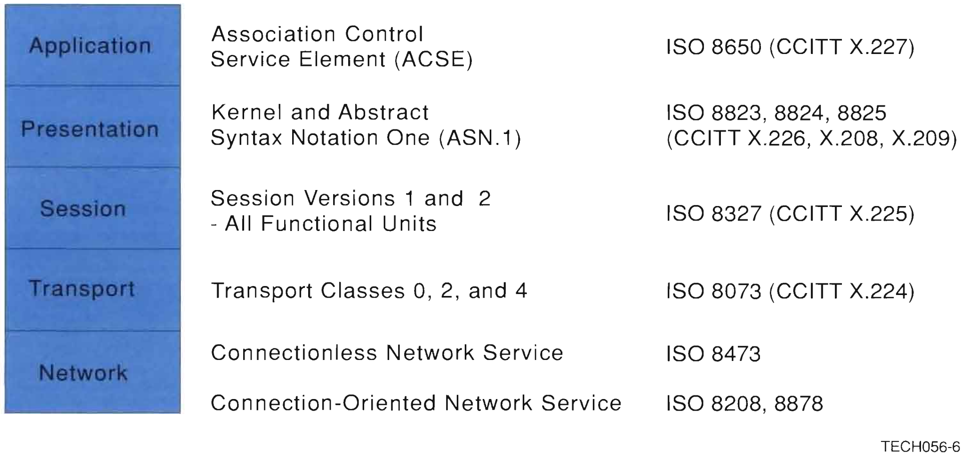
To simplify high-level-language OSI application programming, an Abstract Syntax Checker (ASC) is provided. The ASC is implemented as a control language (CL) command, and it is intended for customer programs that use the programming interface to the presentation layer. The ASC translates data types defined using Abstract Syntax Notation One (ASN.1) into COBOL or C data structures that are included in an application program. This gives application programmers the capability of concentrating on application processing instead of on a particular system's internal data representation.

OSI Layer Protocols Supported

The OSI Communications Subsystem/400 licensed program supports layers 3 through 6 of the OSI reference model, as well as the layer 7 ACSE standard. The OSI layer protocols supported are shown in Figure 43. The Operating System/400 (OS/400) licensed program provides layers 1 and 2 (physical and data link). The OS/400 communications support is structured so these layers can be shared by multiple protocols concurrently (OSI, Transmission Control Protocol/Internet Protocol [TCP/IP], and Systems Network Architecture [SNA]).

The **application layer** (layer 7) manages communications among user application programs and OSI applications. Fundamental to application program control and management are services of the association control service element (ACSE). Support for ACSE is provided as defined in ISO 8650. This support enables two OSI programs in different open systems to:

- Establish or reject an association
- Release such an association in either:
  - Normal mode
  - Abnormal mode



Application	Association Control Service Element (ACSE)	ISO 8650 (CCITT X.227)
Presentation	Kernel and Abstract Syntax Notation One (ASN.1)	ISO 8823, 8824, 8825 (CCITT X.226, X.208, X.209)
Session	Session Versions 1 and 2 - All Functional Units	ISO 8327 (CCITT X.225)
Transport	Transport Classes 0, 2, and 4	ISO 8073 (CCITT X.224)
Network	Connectionless Network Service	ISO 8473
	Connection-Oriented Network Service	ISO 8208, 8878

TECH056-6

Figure 43. *OSI Layer Protocols Supported*

The **presentation layer** (layer 6) handles the data formats of different vendor's systems. It converts the data from one system into a format independent of any vendor's system and reconverts it at the other end to match the receiving system's format.

OSI Communications Subsystem/400 supports the kernel functional unit as defined in ISO 8823, along with support for encoding to and decoding from:

- The abstract syntax used for data definition on open systems as defined in ASN.1

  The licensed program supports the basic encoding rules for ASN.1 as defined in ISO 8825 and ISO 8824 (except for recursion, macro definitions, external references, and value assignments).

- The data structures specified in COBOL or C customer programs

The encoding and decoding support lets the COBOL or C language program be transparent to the actual OSI data transfer syntax.

The **session layer** (layer 5) structures the dialogue between two nodes. It defines such concepts as major and minor synchronization points so that communications can be effectively resumed after a connection or line outage.

OSI Communications Subsystem/400 supports all functional units defined in ISO 8327 DAD (Draft Addendum) 2, Versions 1 and 2. Session Version 2 is based on *Stable Implementation Agreements for OSI Protocols*, Version 2, Edition 1 (December 1988) as described by the National Institute of Standards and Technology (formerly known as the National Bureau of Standards).

The **transport layer** (layer 4) is for end-to-end communications. It provides a data channel between open systems and controls the communications session when the data path is established. OSI Communications Subsystem/400 supports transport classes 0, 2, and 4 as defined in ISO 8073. This includes transport layer splitting and multiplexing. **Splitting** lets a single transport connection run over multiple network connections. The transport connection stays active as long as one or more of the network connections is active. **Multiplexing** lets multiple transport connections share a single network connection, providing for optimal communications resource use.

The **network layer** (layer 3) establishes, maintains, and ends communications between two nodes in the network. The network layer entity of the OSI Communications Subsystem/400 provides connectivity over X.25 subnetworks using connection-oriented (connection setup required) or connectionless (no connection setup required) network protocols.

Configuration

OSI Communications Subsystem/400 maintains a database where all of the information necessary to perform communications is kept. This information consists of a description of the network, specific values used by the OSI layers, and information about applications, directory services, and management services.

The ISO standards provide a great deal of flexibility. The amount of configuration data required to perform meaningful work can be large. This is further complicated by the fact that the data is quite complex, containing many interdependences. This makes configuration difficult because the user must provide most of the data.

The user configures OSI on the AS/400 system using an object-based model that is command driven. With this model, the configuration data consists of 21 configuration object classes. The term object[2] describes a specific instance of a particular object class. An **object** is simply a grouping of data items called attributes. Objects of the same class have the same attributes. The object classes provide a logical, modular view of the configuration data. Figure 44 on page 84 shows the object class hierarchy for OSI Communications Subsystem/400.

The object classes are:

- Abstract Syntax (ABSN): Used to identify abstract syntaxes that the presentation layer encodes and decodes, to or from transfer syntax from or to local syntax.
- Adjacent Node (ADJN): Used to define nodes that are connected to the same subnetworks (adjacent nodes) as the node being configured.
- Agent Registration (AGTR): Used to identify remote nodes (agents) that are managed using the network management commands.
- Application Entity (APPE): Used to define local (at this node) and remote (at other nodes) application entities. An **application entity** is the part of an OSI application that handles communications.
- Application Context (APPX): Used to define application contexts required by applications.
- Application Mode (APPM): Used to define presentation and session layer services to be used on application associations.
- Authority Nickname (AUNN): Used to identify the superior parts (relative distinguished names) of a multiple-part distinguished name. Allows the superior part, which may be shared by many OSI objects, to be specified only once.
- Connectionless Mode Path Set (CLPS): Used to define a set of paths for reaching an adjacent node using connectionless network service.
- Connection Mode Path Set (CMPS): Used to define a set of paths for reaching an adjacent node using connection-oriented network service.
- Directory User Registration (DUAR): Used to identify the directory user agents (DUAs) that are allowed to use the services of the local directory system agent (DSA).
- Inbound X.25 Attributes (IX25): Used to define characteristics of inbound X.25 communications.

[2] The OSI objects are stored in database files.

- Local Attributes (LCLA): Used to define attributes of the local node.
- Line (LINE): Used to define X.25 lines at the local node.
- Line Set (LINS): Used to identify groups of lines. Lines are grouped together for the purposes of starting, stopping, and controlling operations.
- Manager Registration (MGRR): Used to identify remote nodes that can manage the local node.
- Network Service Access Point Address (NSAP): Used to specify local and remote network addresses and their attributes.
- Outbound X.25 Attributes (OX25): Used to define characteristics of outbound X.25 communications.
- X.25 Quality of Service (QOSM): Used to define end-to-end quality-of-service parameters for X.25 communications.
- OSI Route (RTE): Used to define routes for reaching destination nodes.
- Local SAP Selector (SSEL): Used to define service access points (SAPs) and their selector addresses at the local node.
- Subnetwork (SUBN): Used to define subnetworks to which the local node is attached.
- Transport Mode (TPTM): Used to define transport layer services to be used on transport connections.

The interdependences between the object classes follow a hierarchical model. Objects higher in the hierarchy refer to objects lower in

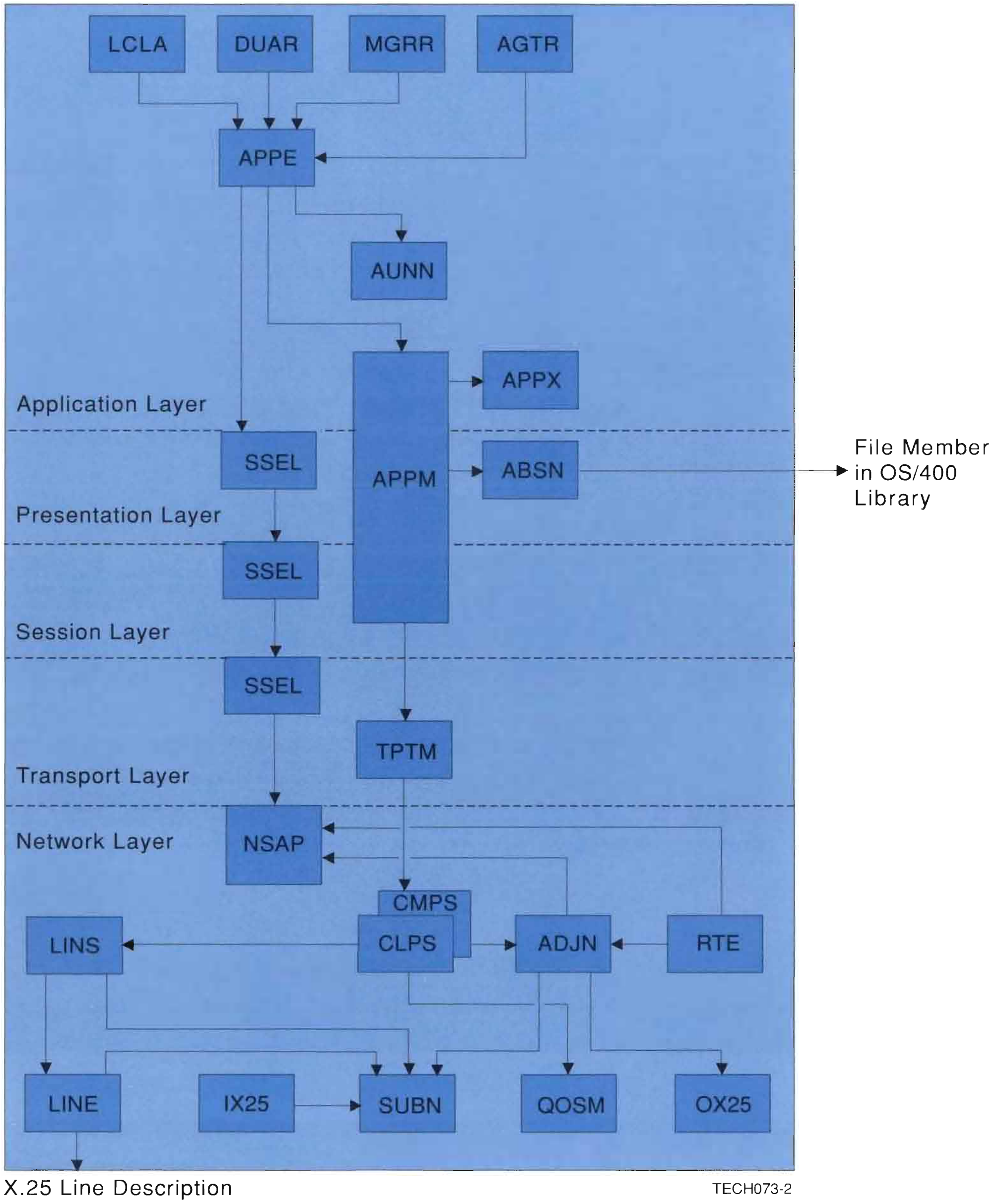


Figure 44. *OSI Configuration Object Classes*

the hierarchy. This means that the higher-level object classes have attributes that are names of specific lower-level objects. In this way the object classes serve as building blocks for configuring the system. The hierarchical model facilitates the sharing of configuration data by allowing more than one higher-level object to point to the same lower-level object.

The user configures OSI with a set of CL commands that correspond to the object classes. There are five types of commands: add, change, display, remove, and work with.... Generally, a separate add, change, display, remove, and work with... command exists for each object class. The command parameters correspond to the object class attributes.

When the commands are processed, validation is done based on the object dependencies. For example, a lower-level object must have already been defined in the configuration database before a higher-level object can refer to it. Additionally, a lower-level object cannot be removed if one or more higher-level objects refer to it. The commands enforce these rules so that the user cannot introduce inconsistencies between the objects.

Network Management

OSI Communications Subsystem/400 allows open systems to report errors and other information so that a network operator can respond accordingly. The management support is based on the Manufacturing Automation Protocol (MAP) 3.0 specification dated 1 August 1988.

The management protocols are derived from:

- ISO Management Framework, Draft International Standard (DIS) 7498-4
- The draft proposal (DP) as defined by MAP 3.0 for:
  - Common management information services (CMIS), DP 9595-2
  - Common management information protocol (CMIP), DP 9596-2

These standards define the concepts of a managing process (or manager) and an agent process (or agent). An **agent** reports errors and information, and processes commands so that its OSI resources can be controlled by a manager. A **manager** sends commands to the agent and receives event reports from the agent. The manager can be local or remote from the agent. The manager and agent are OSI applications known as **systems management application entities (SMAEs)**. A manager communicates with its agent through the common management information protocol (CMIP). The AS/400 system can function as a manager, an agent, or both a manager and agent. Figure 45 on page 86 illustrates the relationship between a manager, its agents, and the use of CMIP.

OSI Communications Subsystem/400 supports the following management services:

- *Get*: Get information about layer attributes, operational parameters, or statistics
- *Set*: Change layer attributes or operational parameters
- *Action*: Request that a specific action be performed by the local resource manager of the layer
- *Event Report*: Notify a manager that a significant event, such as an error or threshold condition, has occurred

Examples of attributes for a specific object class include:

- Threshold events
  - Rejects received
  - Rejects sent
  - Aborts received
  - Protocol data units received and refused
- Protocol violations
- Error counters
- Individual connections
  - Octet (byte) count
  - Message counts
  - Start and stop time
  - Connection-end reason
  - Connection-end events

The AS/400 system in a manager role sends commands to and receives event reports from agents. CL commands can be issued either locally or remotely to perform the required management functions. OSI event reports received from remote OSI agents are converted into SNA alerts, logged in the alerts log, and forwarded to the SNA focal point. The SNA focal point monitors activity on both SNA and OSI networks concurrently.

X.500 Directory

OSI Communication Subsystem/400 provides directory services through an implementation of a subset of the X.500 standards (ISO 9594). This subset is based on MAP 3.0 and partially on the National Institute of Standards and

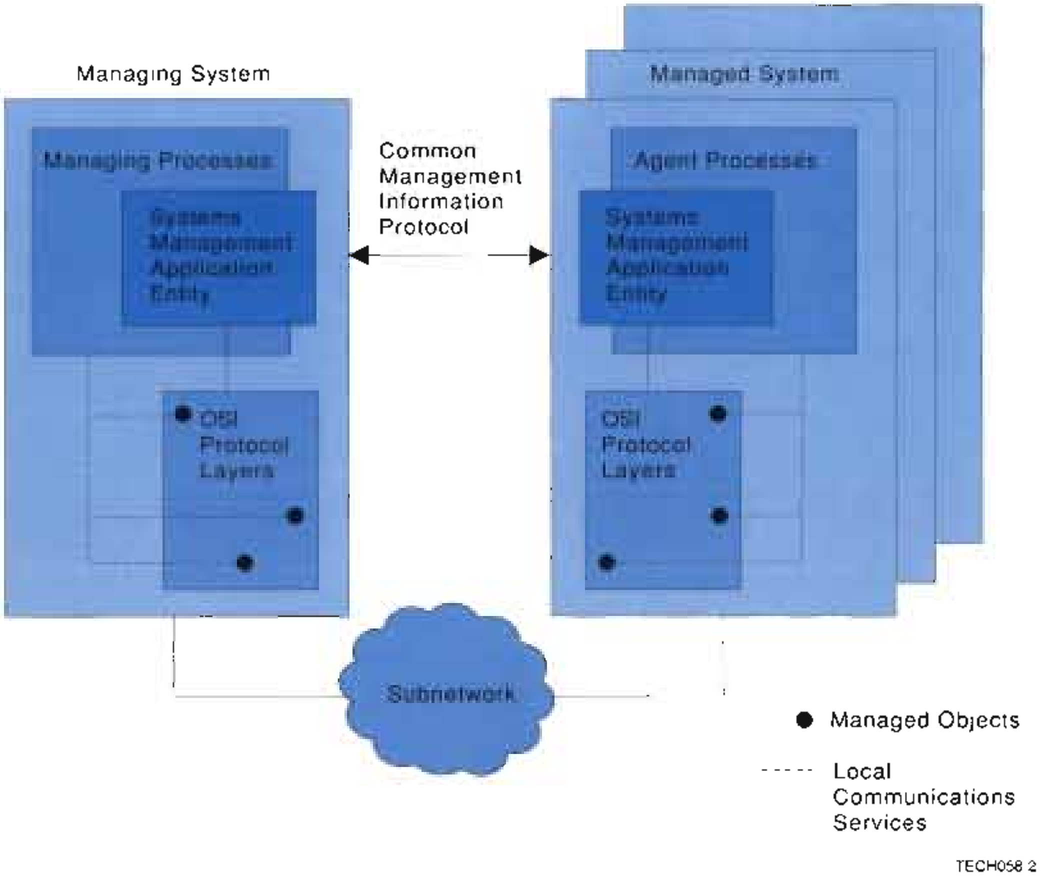

Managing System
Managing Processes
Systems Management Application Entity
OSI Protocol Layers
Common Management Information Protocol
Managed System
Agent Processes
Systems Management Application Entity
OSI Protocol Layers
Subnetwork
● Managed Objects
----- Local Communications Services
TECH058 2

Figure 45. *OSI Management Framework*

Technology OSI Implementation Agreements (December 1988). Both the directory system agent (DSA) and the directory user agent (DUA) are supported as defined in the X.500 standards. An AS/400 system can be either a DSA or a DUA.

A **directory system agent** maintains and controls the directory. Whenever an entity (for example, a user program) requires information from the directory, its **directory user agent** requests and receives information from the directory system agent. The directory access protocol (DAP) is used for communications between the user agent and the system agent.

OSI Communication Subsystem/400 supports a centralized directory service where there is one directory system agent serving multiple directory user agents. The directory is used for application-name-to-application-address mapping. The type of directory is set up using the configuration commands, and is accessed during application entity configuration and when applications using the programming interface need to establish communications with a remote system.

Conclusion

OSI describes the rules and procedures under which equipment from multiple vendors can communicate. When confronted with a current (or potential) multiple-vendor environment, an organization can use OSI as a basis for vendor-independent electronic information exchange. The OSI Communications Subsystem/400 licensed program enables the AS/400 system to participate in this environment.

Network Openness through User-Defined Communications

Describes the functions of the AS/400 user-defined communications support, which provides an interface to system communications functions.

Sarah R. Jones

Introduction

The expanding AS/400 communications environment requires the capability to connect to a diverse range of systems and devices by means of existing and emerging telecommunications protocols. The user-defined communications support of the AS/400 system facilitates the implementation of virtually any telecommunications protocol. To accomplish this, the support provides an interface to the data link control layer, as defined by the International Organization for Standardization (ISO) in the open systems interconnection (OSI) reference model. In effect, this support provides a level of flexibility for an AS/400 customer or business partner to develop communications protocols, especially those not provided by IBM, through AS/400 product offerings.

User-defined communications support provides an interface to common communications protocols used at the data link control layer. This interface is designed to satisfy objectives, such as openness to data link control functions, protocol flexibility, and coexistence among communications protocols sharing the AS/400 system and the physical network.

Additionally, the communications support is integrated within the communications functions of the Operating System/400 (OS/400) licensed program.

Overview of Function

To clearly illustrate the capabilities provided by user-defined communications support, a comparison is made to the OSI reference model. Figure 46 describes the relationship of the user-defined communications application within the framework of this model.

In Figure 46, the user-defined communications support is the OS/400 program interface between the user-defined communications application program and the data link layer. The layers that exist above the data link layer are implemented by the user-defined communications application program. The functions provided by layers above the data link layer are defined in the user-defined communications application.

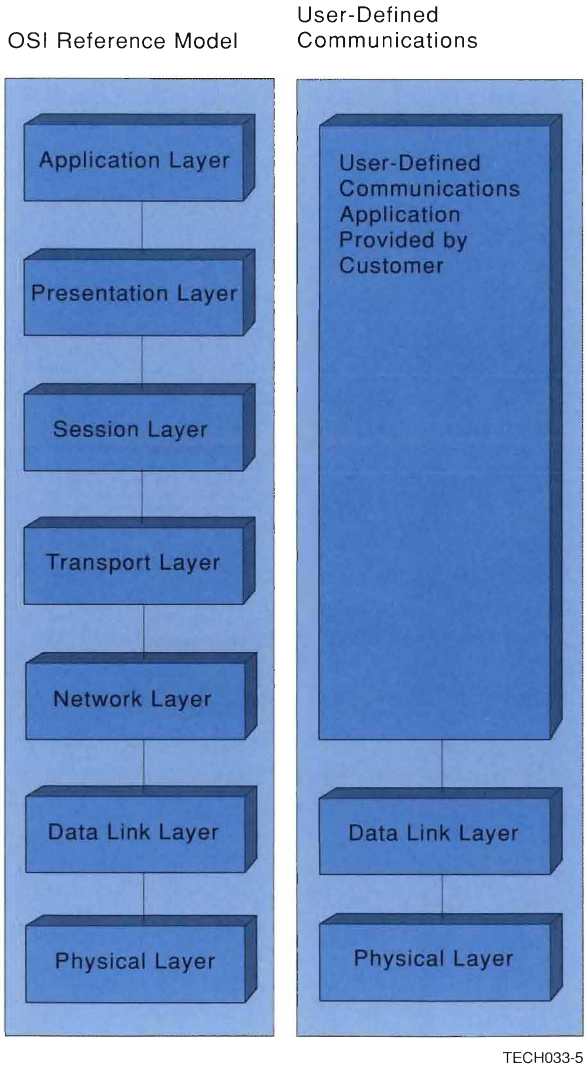


Figure 46. *User-Defined Communications Comparison to the OSI Reference Model*

The data link control functions to which the user-defined communications support provides access consist of protocols used in wide area networks (WANs) and local area networks (LANs). The WAN support provides application access to the packet-switching protocols at the International Telegraph and Telephone Consultative Committee (CCITT) X.25 layer 3. The LAN support provides access to protocols that use the Institute of Electrical and Electronics Engineers (IEEE) 802.2 logical link control (LLC) interface or Ethernet Version 2.

Examples of functions allowed through the user-defined communications interface are establishing and ending connectivity to the network, and sending and receiving various types of data frames across the network to and from any number of systems. Additional functions include routing data, obtaining line status information, and using system timers.

User-defined communications support is integrated into the OS/400 licensed program and directly accesses the licensed internal code for the communications support. Figure 47 shows the user-defined communications support, which resides above the machine interface. Below the machine interface is the licensed internal code, which interfaces with the communications input/output processor.

Figure 47 is based on the AS/400 communications structure that is described in the article "Communications and Networking Structure" in *IBM Application System/400 Technology*, SA21-9540. The components of the figure that are not described in the article are Ethernet communications support and the open station input/output (I/O) manager. Ethernet communications is provided by the input/output processor and the line input/output manager in the same way that other data link protocols are supported.

Like other AS/400 station I/O managers, the open station I/O manager is responsible for managing connections and requests from the machine interface to the data link control layer. The open station I/O manager is also responsible for routing data and incoming connection indications to the correct protocol based on routing information from the application program that resides above the machine interface. Routing information is based on standards definitions. Examples of routing information include destination service access point (DSAP) and source service access point (SSAP) for IEEE 802.2 protocols, and protocol identifier and data terminal equipment (DTE) address for X.25. Specifying unique routing information allows multiple user-defined communications application programs to communicate simultaneously to the network.

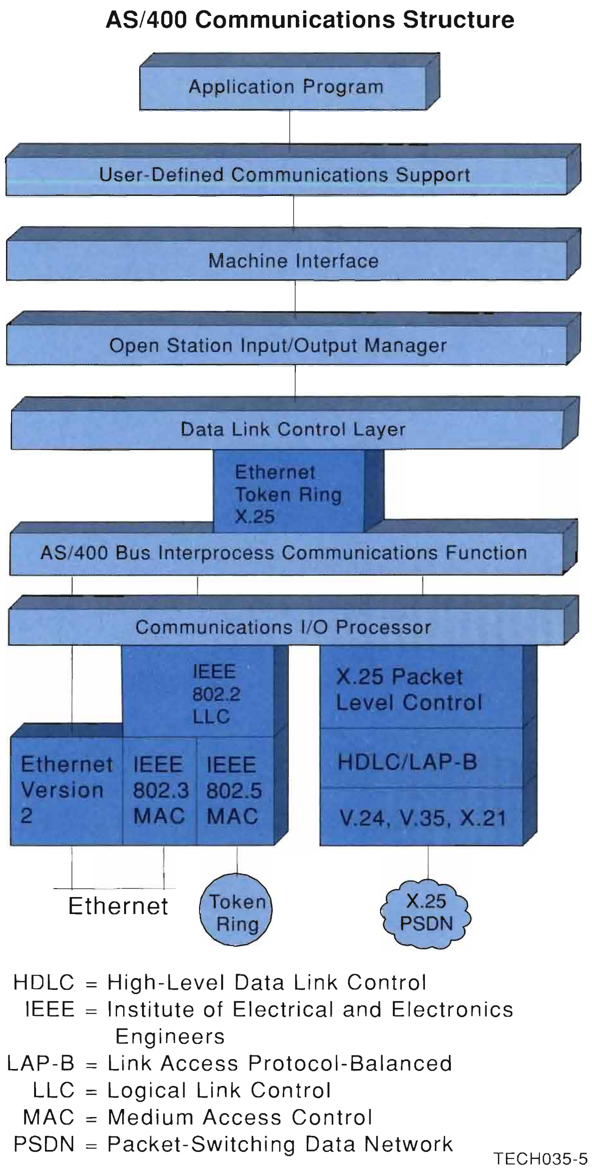

AS/400 Communications Structure
Application Program
User-Defined Communications Support
Machine Interface
Open Station Input/Output Manager
Data Link Control Layer
Ethernet
Token Ring
X.25
AS/400 Bus Interprocess Communications Function
Communications I/O Processor
IEEE 802.2 LLC
X.25 Packet Level Control
Ethernet Version 2
IEEE 802.3 MAC
IEEE 802.5 MAC
HDLC/LAP-B
V.24, V.35, X.21
Ethernet
Token Ring
X.25 PSDN
HDLC = High-Level Data Link Control
IEEE = Institute of Electrical and Electronics Engineers
LAP-B = Link Access Protocol-Balanced
LLC = Logical Link Control
MAC = Medium Access Control
PSDN = Packet-Switching Data Network
TECH035-5

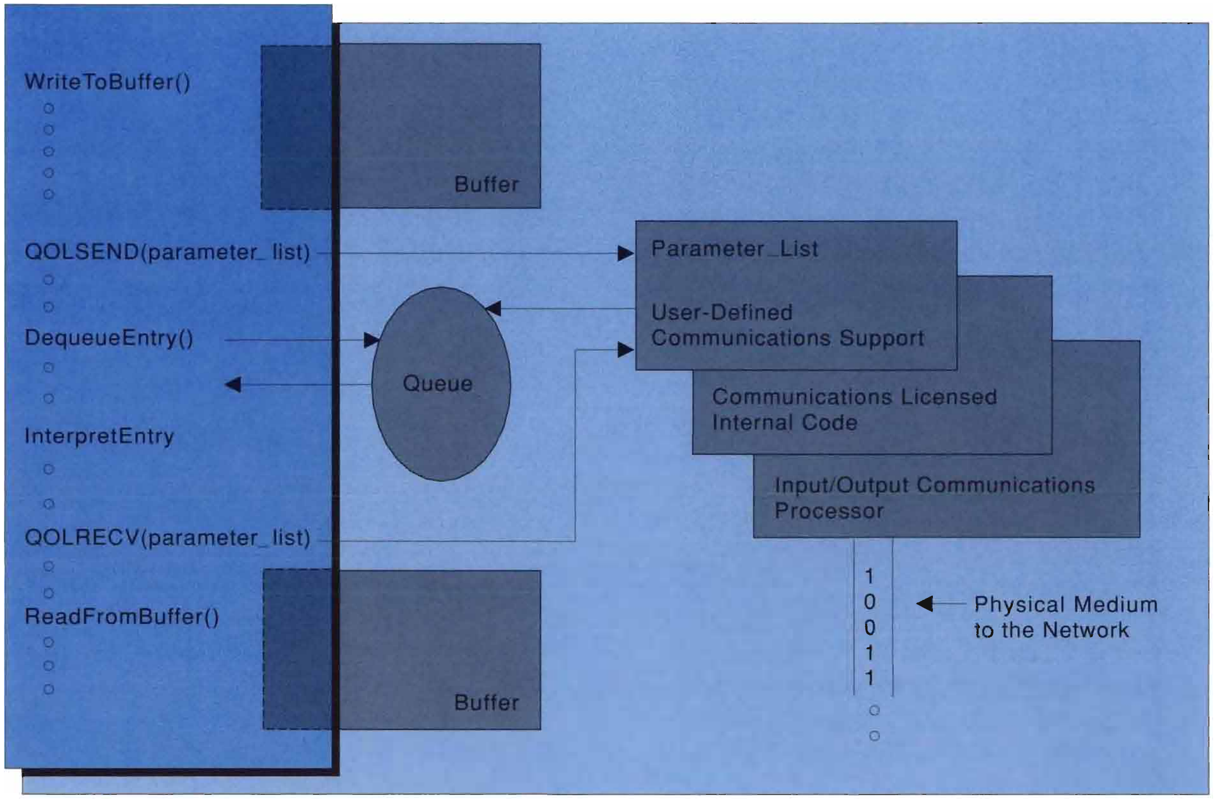
Figure 47. *User-Defined Communications within AS/400 Communications Structure*

Application Interface

A user-defined communications application gains access to the communications support by use of routines called by a program. The routines provide a pipeline into the network for the application, which can be a simple program or a complex protocol containing many well-defined layers involving many programs. The use of the routines is consistent across both LAN and WAN networks, where only the data passed through the buffers is tailored for transportation across the network type.

The constructs for communicating between the application and the communications support are the queue, the data buffer and descriptor spaces, and the parameter list.

Figure 48 shows the relationship of input and output buffers, the queue, user-defined communications support, and licensed internal code to the user-defined communications application program. On the left of the figure, pseudocode represents a program flow of routine calls. The user-defined communications routines shown, QOLSEND and QOLRECV, represent the output and input functions. The remaining routines represent functions that are performed by the application. The constructs in the center of the figure are the common interface objects used for intra-process communications between the application program, which is depicted on the left side of the figure, and the user-defined communications support, which is depicted on the right side of the figure. The constructs to which both the application and the user-defined communications support have direct access are the data buffers, the queue, and the parameter list.



TECH034-3

Figure 48. *Application Interfaces and Constructs*

The input and output data buffers contain information for data frames and call control frames, which are transmitted across the network. These buffers have defined formats, which correspond to the different types of frames being transmitted and received.

The queue provides a mechanism for the user-defined communications support to notify the application of asynchronous events. Examples of these events include notification of inbound data and connection failures.

The parameter list is used to indicate the operation or request and any additional information the user-defined communications support needs to perform the function. The success or failure of requested functions is reported through the parameter list.

Network Scenario

The data link layer interface enables system programmers to provide communications protocols and network options that are otherwise not provided. To provide new protocols and network services, the communications application is responsible for providing the functions and services required by the protocols above the data link layer.

In a simple network scenario, the user-defined communications application implements a protocol over a single network, connecting the AS/400 system with the other systems on the network. A more complex scenario includes multiple networks and multiple network types. In the more complex scenario, the application has simultaneous connectivity with multiple networks, thus expanding the internetworking possibility of the protocol implemented by the application and the AS/400 system.

Figure 49 shows multiple, independent networks interconnected through the user-defined communications application implemented on both AS/400 systems. The figure shows how the user-defined communications support enables an application program to provide a variety of connectivity options and services. The application can communicate with one or many of the remote systems simultaneously and can provide routing services between the three networks (token ring, X.25, and Ethernet).

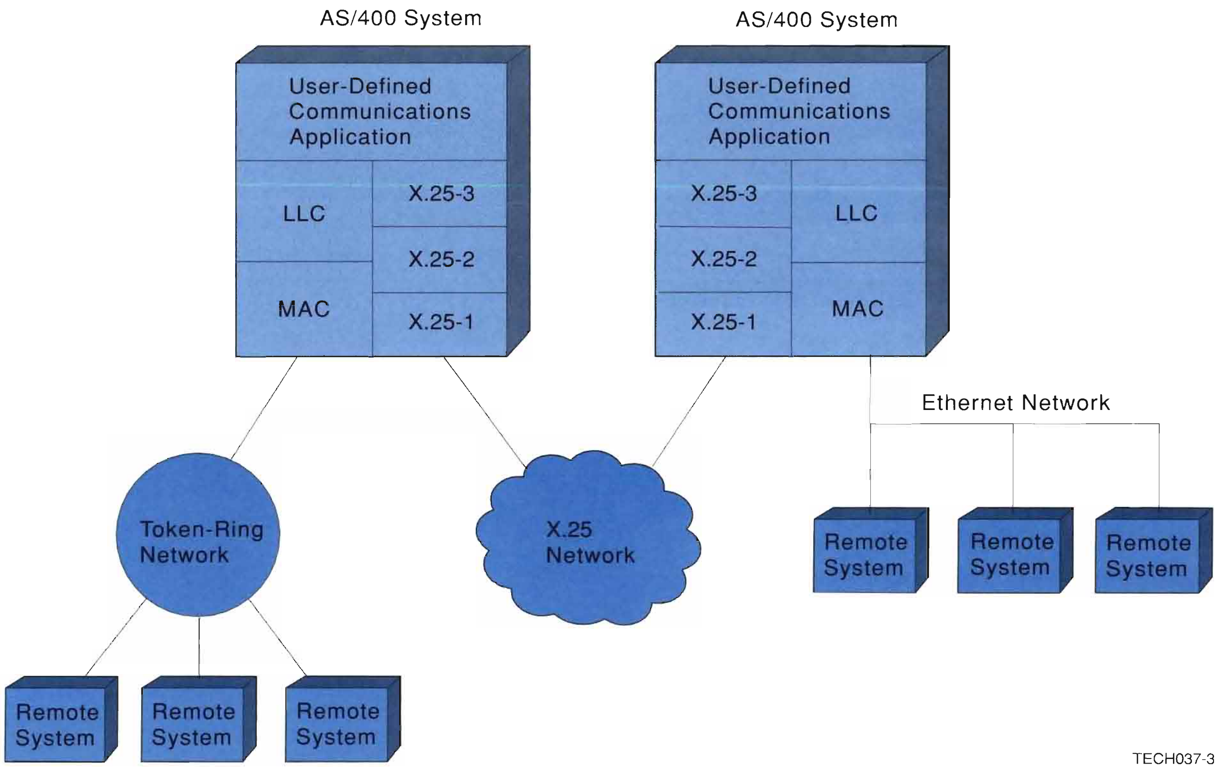


Figure 49. *User-Defined Communications Application in a Complex Network*

Conclusion

Increased focus on interoperability and openness encourages the offering of application enablers that facilitate the capability to connect dissimilar systems across dissimilar networks. User-defined communications support provides the customer and IBM business partners with the opportunity to include the AS/400 system in heterogeneous environments where it was previously not possible.

Integrated Services Digital Network

Discusses the AS/400 system's integrated support for integrated services digital network (ISDN).

David G. Carlson, James J. Herring, and
Brian E. Jongekryg

Introduction

Integrated services digital network (ISDN) is an emerging new technology in the field of data and voice communications. It provides a high-speed, relatively error-free transmission medium for data communications as well as advanced call-related functions, for example, calling party identification.

Version 2 of the AS/400 system provides support for the direct attachment to an ISDN through an integrated adapter, allowing both existing and new data communications applications to use the ISDN. This application connectivity is accomplished with the aid of a newly architected data link communications protocol known as **ISDN data link control (IDLC)**. IDLC, a fully balanced protocol,[3] is well suited for the duplex ISDN environment. Finally, the system uses call information provided by ISDN to help ensure security and compatibility when connecting to adjacent systems and work stations.

ISDN: The Digital Highway

Information is the fundamental part of every business, laboratory, and university in existence today. This information exists in many forms, including FAX, image. data, and voice. Currently, many different technologies and media exist for the transportation of this information between two parties. Recognizing the need for a common, fast, and architected mechanism for information movement, the International Telegraph and Telephone Consultative Committee (CCITT)[4] defined a set of standards for the interface to ISDNs.

A key design point that the CCITT adhered to during the definition of the ISDN interface was that with the currently available technologies, twisted pair wire is able to transmit information at about 200,000 bps. A large percentage of existing telephone network wiring is twisted pair. However, because analog signaling is used, the transmission of encoded digital information is typically limited to 2,400 bps. The ISDN standards define the interface to a high-speed, information independent, digital network such that connectivity can be accomplished using existing wiring. Another function of ISDN is that the network performs "out-of-band" call-setup procedures on a special channel called the **D-channel**. This function can be contrasted with telephone signaling, where call control procedures and user information share the same medium at different times. For example, the user sets up the call using audible tones or pulses, the network places the call and dedicates the medium to passing user data, for example, voice. If signaling is required while in conversation, a special tone must be generated, and then call control procedures can be performed.

The separate D-channel for network signaling allows for fast, simultaneous (with user data) call control, such as conferencing or routing. The CCITT defined a high-level, message-based protocol for use on this signaling channel. This architected protocol, called **Q.931**, allows for the transfer of information during the call establishment phase, such as presentation of the calling party number to the called user. These types of functions are not unique to ISDN. However, ISDN provides an architected mechanism to perform the functions so that equipment manufacturers do not have to contend with multiple proprietary implementations.

[3] Communicating partners are considered peer, and either partner may send data at will, that is, no polling is required.

[4] An internationally recognized standards organization responsible for many widely used architectures, such as X.25 and X.21.

Realizing that a dedicated channel for call control may result in wasted bandwidth in some environments, the CCITT architected an interface to an integrated packet handler (X.25) over the D-channel as well. This X.25 interface allows the same ISDN interface to support not only the telephone network type of functions, but also integrated access to a packet-switching data network (PSDN) simultaneously.

User information is carried to the ISDN on bearer channels, or **B-channels**, each providing a 64,000-bps duplex, digital connection to the adjacent party. For B-channels, the CCITT architected only the physical transport or layer 1. The format of the higher layer information on the B-channels is not defined by the CCITT. For example, two users can exchange voice, FAX, video, or user information encapsulated in data communications protocol frames. The CCITT defines two types of ISDN interfaces, differentiated by the number of B-channels supported:

- Basic rate interface (BRI)

  Two B-channels plus one 16,000-bps D-channel are supported with this interface. A unique function of the BRI is the ability to attach up to eight different devices to the same interface, forming what the CCITT refers to as a **passive bus**. The architecture includes a collision detection mechanism, which ensures that contention for the D-channel by the devices is arbitrated properly. The network assigns the B-channel to a device according to the D-channel setup procedures and dedicates its use to that device until it is disconnected.

  Thus, on a single BRI, a user can attach multiple devices, such as telephones, FAX machines, or computers in any combination.

- Primary rate interface (PRI)

  This interface supports 23 (or 30 in Asia and Europe) B-channels plus one 64,000-bps D-channel. A unique function of the PRI is the ability to bundle B-channels into higher bandwidth **H-channels**. For example, an H0-channel is made up of six B-channels, resulting in a 384,000-bps connection.

See Figure 50 on page 94 for an illustration of a conceptual ISDN configuration. Notice that D-channels end in the ISDN switching equipment adjacent to the attached terminal equipment. Only B-channels have end-to-end connectivity through an ISDN.

ISDN is an evolving standard, which supports multiple information types on a single interface, and can be implemented over existing telephone networks. The key point above is *evolving*. The set of standards were defined over several years and many vendors began their implementation without a firm architecture. The result is that some ISDN switch equipment, which is claimed to be CCITT compliant, may actually deviate greatly from the standard. However, now that a majority of the architecture is complete, standardized equipment is becoming more readily available, resulting in more ISDN connectivity options.

Design Objectives

Several factors were considered when designing the software to support ISDN on the AS/400 system. Characteristics, such as ease of use, network compatibility, and operating system principles, were refined into a set of design objectives. These were used to guide the design and later the implementation and testing of the software.

Integration

The AS/400 system has consistently integrated new technology into its machine architecture and operating system. A key design objective was to ensure that existing customer applications take advantage of the ISDN functions with minimal effect on configuration. This required that the support be embedded in the base object structure and task model.

Flexibility

One of the key attributes in any successful design is the ability to adapt to new requirements or changes in technology. Because network implementations of the ISDN standards are still evolving, flexibility becomes one of the overriding factors in the system's ISDN implementation. For example, one of the many different network provider offerings includes a 1B+D subscription. That is, instead of providing a true BRI, networks are providing clients having data-only requirements, an option where they use only one 64,000-bps B-channel at a reduced cost. Although a simple example, it illustrates the level of flexibility required in the design to accommodate

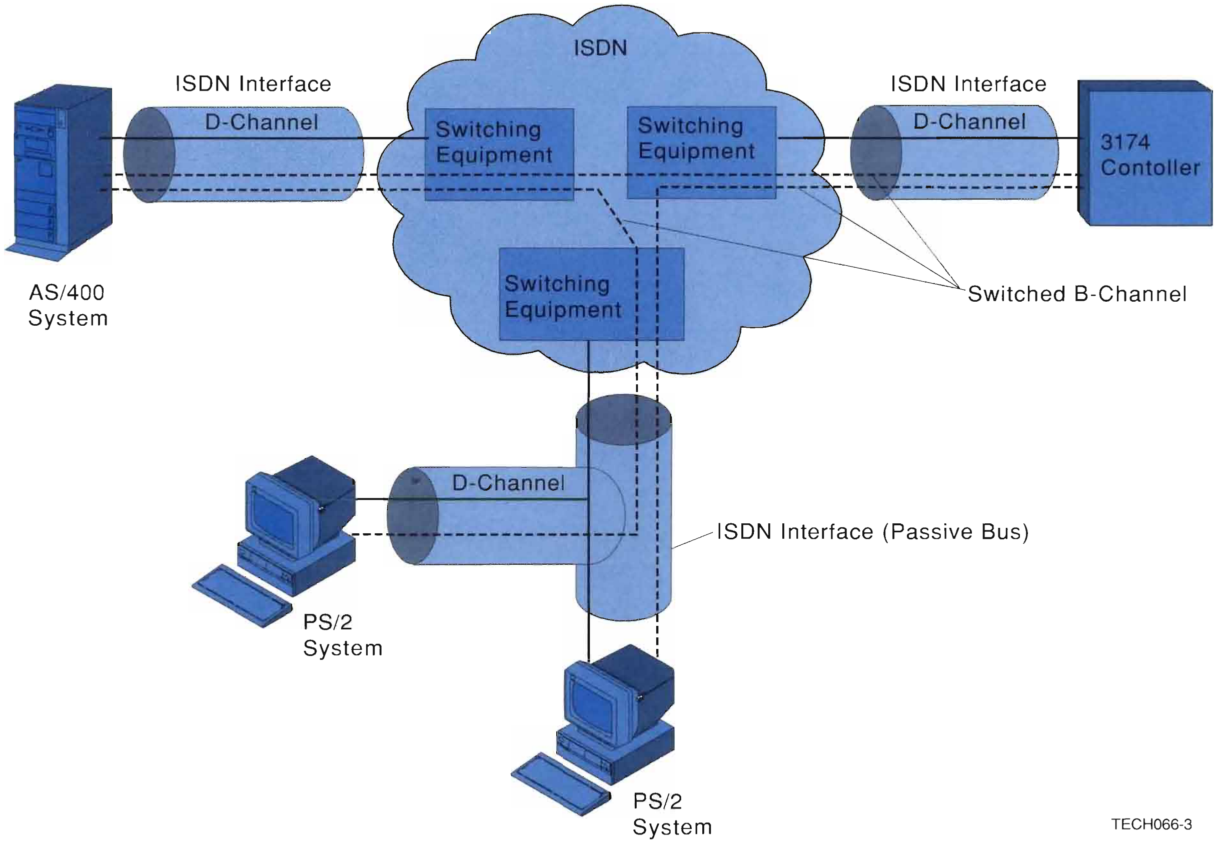


Figure 50. *ISDN Connectivity*

the differences between network service offerings. The system's flexible ISDN design allows a user to easily take advantage of the diversity in offerings.

Simplifying Complexity

Complexity traditionally accompanies flexibility. Every option in connectivity and configuration challenges users to make the right choice. One of the central issues in designing the system's ISDN support was reducing complexity by supplying inexperienced users with the right choice using global defaults while still providing more knowledgeable users with configuration parameters that could be tailored to fit their unique network and connectivity requirements.

Future Considerations

Finally, with ISDN still in its infancy, it is difficult to predict what services customers will require and when networks will provide those services. A requirement for any ISDN implementation is that it can be extended to support new connectivity functions, such as X.25, additional services, such as network management, and higher bandwidths, such as primary rate and broadband ISDN.

Object Structure

One of the key elements of the AS/400 ISDN implementation is its integration into and extension of the AS/400 communications object model.

Object-Based Data Communications Model

Most data communications implementations require that the user configure elements that describe the physical hardware and protocols required for connectivity. AS/400 communications configurations are described by a set of objects, organized by the type and scope of protocol or interface being used. In the Version 1 AS/400 system, the key objects in the structure were the line description, controller description, and device description.

- Line description

  The line description describes both the physical interface to the network, or transport medium, and the associated *lower layer protocol stack*[5] required for communications with the adjacent network or communications equipment, such as synchronous data link control (SDLC), X.25, or token ring.

- Controller description

  The controller description describes the adjacent system, controller, or work station, that is, the communications partner. It further defines the *higher layer protocol stack*, such as advanced program-to-program communications (APPC), to use when communicating with the remote equipment. The SNA type of controller descriptions include APPC and APPN (LU 6.2), host (LU 2) and work station (LU 7 for example). Non-SNA controller descriptions include the network type used by open systems interconnection (OSI), Transmission Control Protocol/Internet Protocol (TCP/IP), and user-defined protocol stacks, as well as asynchronous and bisynchronous types.

- Device description

  The device description describes much of the addressing and session characteristics required by the particular higher layer protocol stack used, such as SNA logical unit addressing and request unit sizes.

These objects are arranged in a hierarchy. Device descriptions are attached to controller descriptions, which, in turn, can be attached to line descriptions as illustrated in the darker shaded portion of Figure 51. Notice that controller descriptions representing switched connections, such as switched lines or X.25 switched virtual circuits, may refer to several line descriptions on which to accept or initiate calls. Controller descriptions that represent nonswitched connections, such as leased lines or X.25 permanent virtual circuits, only refer to one line description representing the nonswitched transmission medium.

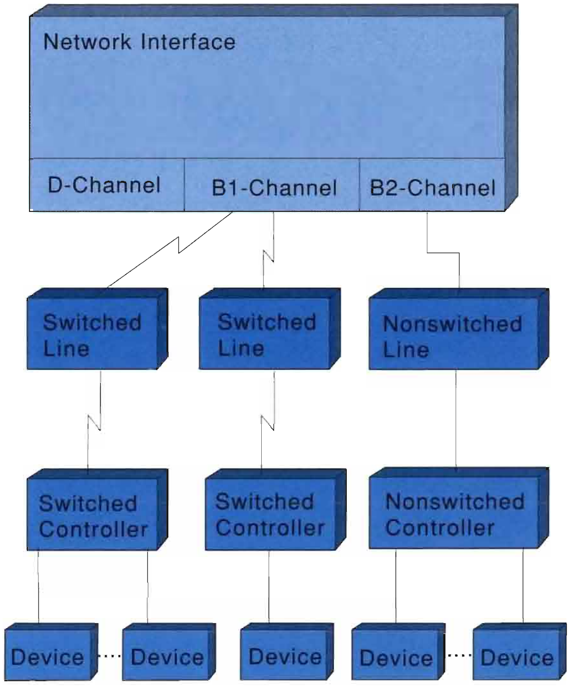


Figure 51. *AS/400 Communications Object Hierarchy*

See the *Communications: Operating System/400* Communications Configuration Reference*, SC41-0001, for a more detailed description of these objects.

Extending the Hierarchy

Several types of line descriptions represent lower layer protocol stacks that support the multiplexing of logical connections on a single physical medium, such as X.25 (logical channels) or token ring (service access points). For example, each of the several virtual circuits supported by X.25 lines may support different *higher layer protocol stacks*, each represented by a unique controller description. However, in the case of ISDN, each of the B-channels may actually support unique *lower layer protocol stacks* concurrently, for example, X.25 on one channel and IDLC on another. The pre-Version 2 object model did not include the ability to multiplex several lower layer protocols over one physical interface.

To support this type of multiplexing, a new object called the network interface description was introduced. This object allows the one physical interface to support multiple lower layer protocol stacks (line descriptions) simultaneously. The result is that the object hierarchy has grown by one object, the network interface description, to which line descriptions may attach. Figure 51 illustrates how the ISDN network interface descriptions fit into the existing communications object hierarchy. Notice that the figure illustrates how more than

[5] The term protocol stack refers to the layers of software that support the transmission of information between adjacent communicating equipment.

one switched line can reference the switched B1-channel, allowing them to contend for the switched resource. The nonswitched line is dedicated for use on the B2-channel.

Network Interface Description

The network interface description integrates ISDN's multiple, heterogeneous data links into a new communications object. The object describes the various data and voice bearer channels that are controlled by one signaling channel. Version 1 objects effectively handle only one level of complexity. The network interface description combines the dynamic and multiplexing nature of ISDN into one system object and becomes the central representation of an ISDN interface on the system. This network interface description is:

- A configuration object that simplifies the complex world of ISDNs by providing defaults for all but two parameters, the object and resource names. The network type parameter allows the user to specify the network or type of ISDN switch to which the ISDN interface is attached. This allows the system to provide values for many of the parameters affected by network differences. The network type may be defined in the system's network attributes with the ISDN configuration objects referring to that network attribute. Because a user typically works with only one network provider, this allows the user to set the value once and only once.
- A complete representation of the physical resources and channels associated with a customer's ISDN subscription. A single network interface description describes all the interfaces that are controlled by a single D-channel. Each interface's channels are described by a **channel map** within the network interface description. This allows for the multiplexing of *physical resources* with potentially different properties.
- A new level in the existing communications object hierarchy. This provides the multiplexing of *logical resources* within an interface. A channel (or group of channels) is associated with a line description, thus allowing the lines to remain conceptually the same by separating the physical ISDN interface from the lower layer B-channel protocol. The other objects in the communications object hierarchy are unaffected.
- The basis for supporting network-directed resource allocation dynamically. Each entry in the channel map may be dynamically assigned a line description. The system assigns a line to a channel (or group of channels if the network or user specifies an H-channel) dynamically upon notification by the network that a connection has been initiated by remote equipment. Furthermore, based on information supplied by the network, the line with the proper bandwidth capabilities and requested lower layer protocol stack is selected and activated upon receipt of the notification. This allows the system to implement the X.31 and X.32 identification procedures necessary to fully support switched access to X.25 networks through an ISDN interface.

Answer Management

As discussed in the topic "ISDN: The Digital Highway" on page 92, ISDN provides a mechanism for passing data in information elements (such as calling or called party numbers, calling or called party subaddresses, and user-to-user information) from the calling user to the called user. The intent of the architecture is to allow the called party to use the received data when determining whether the incoming call should be accepted or rejected. An analogy could be made to a database query where the information elements passed by the ISDN are like search keys, and the returned result indicates whether the call should be rejected or accepted and the pertinent routing information.

The receipt of call information is similar to that defined in CCITT Recommendation X.25. ISDNs, however, allow only seconds to process incoming calls while X.25 networks allow minutes. The algorithm used by the AS/400 system must have the flexibility to make use of the provided information but still ensure that the call is processed within the network time limits.

A new function exists in the AS/400 system, called answer management, which is responsible for determining if an incoming ISDN call should be accepted by the system. If so, answer management determines which line description should accept the call.

This function uses a new communications configuration object, the connection list, during call processing. The connection list object, the actual algorithm, and significant implementation details are covered in the following subtopics.

Connection List Objects

The connection list object consists of one or more entries, which can be configured by the user to contain ISDN information. The local system passes this information on outgoing calls and also uses this information for screening incoming calls, the latter case being discussed here. The entries contain several different fields, each corresponding to an information element, which may accompany the call. The user can provide information that matches the corresponding received information element data. Other special values can be configured, such as *ANY (any information element data value including none is acceptable) and *NONE (the corresponding information element is not accepted).

For a particular incoming ISDN call, a connection list object that contains an entry with all of its fields matching the received information takes precedence over a connection list object that has even one generic match, that is, a value of *ANY.

The AS/400 system considers line descriptions to be targets for incoming ISDN calls. Therefore, when configuring a line description to be used for answering ISDN calls, one must specify a connection list object to be used for call screening. The system provides a default connection list to generically match the information received with any call, thus allowing any line descriptions referring to this connection list the opportunity to accept any incoming ISDN call.

Figure 52 illustrates the contents of a connection list object and how the other communications objects reference it. Notice that the line description references the entire connection list for answering calls, while the controller description names the single entry to use when placing outgoing calls.

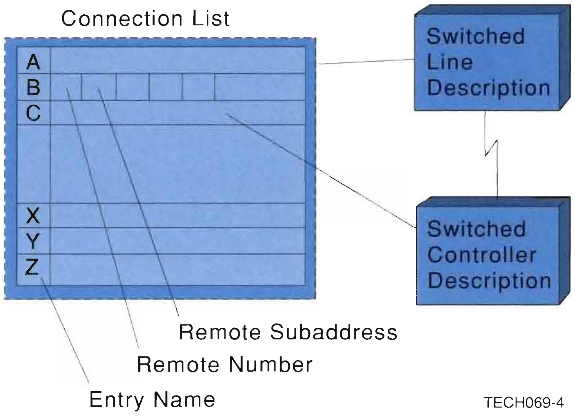


Figure 52. *Connection List Configuration*

ISDN Call Processing Algorithm

The answer management support begins by filtering out calls that request functions not currently supported by the system, such as voice or video. The next phase of call processing parses the received information into key fields, that is, those fields that can be screened by connection list objects.

Next, a count of the greatest number of field matches for a given entry is calculated for each connection list object. A negative count means that no entries match the call, and zero indicates the connection list matches only generically, that is, *ANY is configured for all fields. As these counts are calculated, the system orders the connection list objects by their counts from the most matches to the least.

Finally, for each connection list object on the ordered list, the system trys to isolate a line description that references the object and is able to accept the call. Notice that because the list is ordered, the system gives preference to those connection list objects having the highest number of exact matches. However, if no line descriptions are available for the call, processing continues with lower priority connection list objects and their associated line descriptions.

Implementation

To implement the call processing algorithm directly would likely result in the ISDN call processing time limit being exceeded. For example, all of the required object references would result in excessive page faulting, and linear searches would further degrade performance. To enhance the performance of the algorithm, data from the connection list object entry information is mapped into **machine indexes** (binary radix search trees supported by the AS/400 system). One index exists for each field category supported by the system, such as calling party number or calling party subaddress. The indexes provide the ability to perform searches in logarithmic time. To ensure that the information maintained in the indexes exactly matches the connection list object information, the answer management support is notified whenever changes are made to the objects. To further reduce search time, only entries from active connection lists (at least one line description that references the list must be varied on) are inserted into the indexes.

The search procedure begins by searching the calling party number index, which is the index with the highest likelihood of having unique entries. If a match is found, the resulting index key data identifies the connection list object and entry to which it corresponds. Searches are performed for the remaining fields of that connection list entry. The first field that fails both an exact and a generic search results in the entry being bypassed. The procedure continues with the next matching entry in the calling party number index, if any.

As matching entries are found, an exact match count is calculated and maintained in data structures that represent the connection list objects called connection list source/sink active device lists (SSADL). The SSADL structures are linked together in decreasing magnitude order of their match counts. During processing, many of the SSADL structures must be referenced. To help make page faults more productive, these structures are packed into contiguous storage so that several of them may be retrieved in a single access. Notice that with the algorithm, there is no need to actually reference the connection list object directly, thereby limiting unnecessary page faults.

When no more exact, or finally, generic matches are located in the first index, the answer management support attempts to identify the line descriptions that actually reference the ordered connection list objects. This is done by searching yet another machine index that maps connection list objects to the line descriptions that reference them. Additional information is located in this index, such as line state and protocol type. This information enables line descriptions that are not call candidates to be screened without referencing the line description directly. The address of each eligible line description, located during connection list object processing, is stored in an array, which is dynamically increased to accommodate the line description candidates. To help speed up the actual processing of the line description candidates, a fixed number is asynchronously paged into main storage.

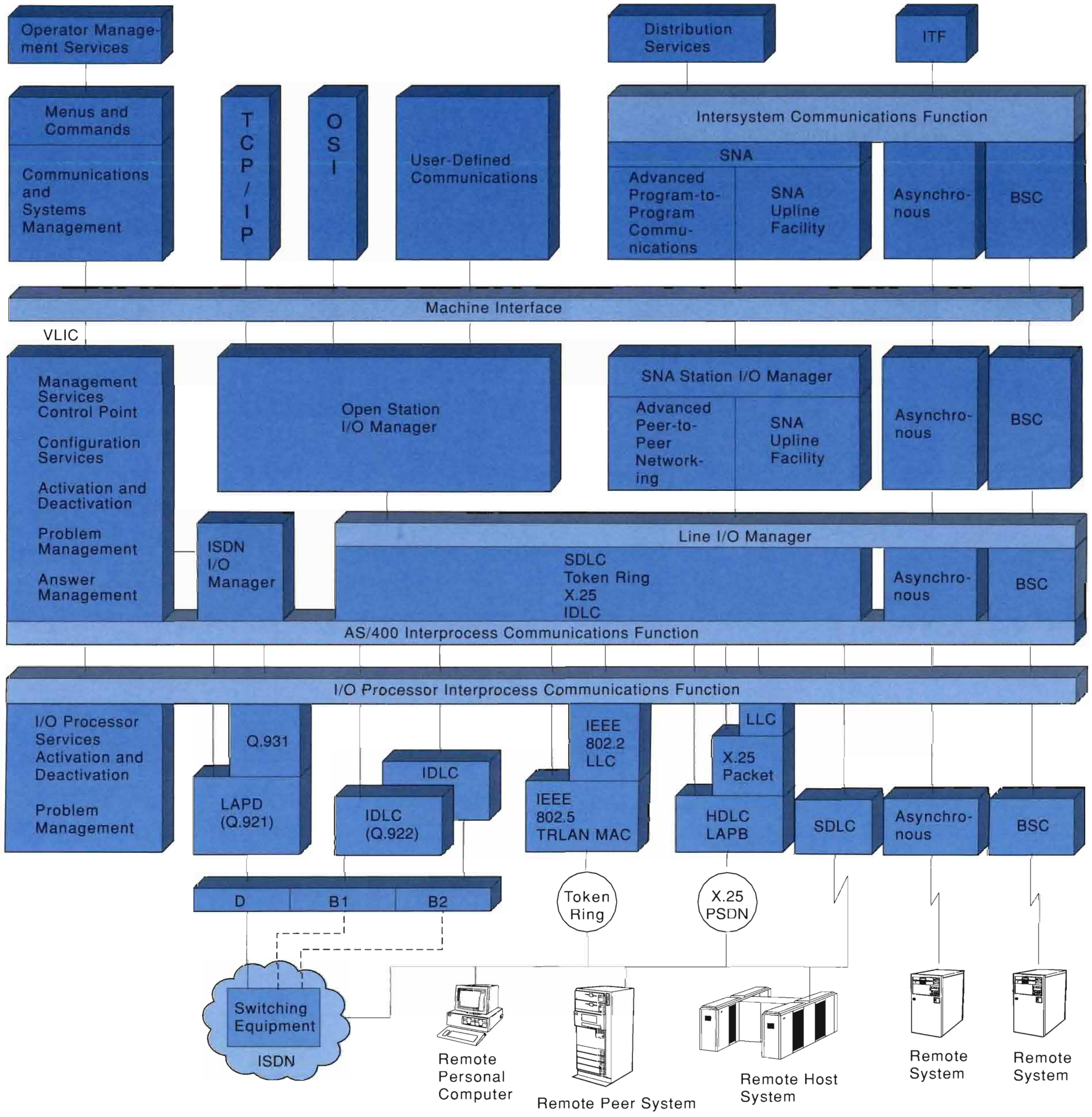
In the final part of the algorithm, each of the line descriptions is interrogated to see if it can accept the incoming call. As a line description is checked and bypassed, a reason code is saved in the array, and another one is asynchronously paged. This ensures that a fixed number of line descriptions remains in main storage during the selection process. Processing stops with the first line description that is able to accept the incoming call.

If no line descriptions are eligible to accept the incoming call, a portion of the dynamic array, containing the names of the line descriptions and reason codes, is saved in the error log and a message is sent to the system operator message queue. If problem analysis is run from the message, it makes use of the information logged by the answer management support to determine why each line description was considered ineligible to receive the call.

The problem analysis procedure also analyzes any user-selected line descriptions to determine why they were not considered. This last function is helpful in the case where the intended line description was not varied on by the user, which means its corresponding connection list object entry information was not even available to the answer management support.

Communications Structure

ISDN support on the AS/400 system takes advantage of the system's existing communications structure (see the article "The Communications and Networking Structure" in *IBM Application System/400 Technology*, SA21-9540). The management services control point maps the existing communications structure to the new ISDN environment and also adds a new function for ISDN, that of answer management. In addition to the management services changes, two new I/O managers were added to the OS/400 task structure. One manages the ISDN network interface and the other manages line descriptions representing the IDLC protocol. Figure 53 on page 99 depicts the integration of those new functions into the system.



BSC	=	Binary Synchronous Communications
HDLC	=	High-Level Data Link Control
IEEE	=	Institute of Electrical and Electronic Engineers
ITF	=	Interactive Terminal Facility
LAPB	=	Link Access Procedures-Balanced
LAPD	=	Link Access Procedures-D-Channel
MAC	=	Medium Access Control
VLIC	=	Vertical Licensed Internal Code

TECH070-7

Figure 53. *AS/400 Data Communications Components*

Management Services Control Point

The management services control point (MSCP) is not a new task for ISDN; however, it provides several new functions in support of ISDN network interfaces and ISDN call processing. Although the job of processing incoming ISDN connections resides in the answer manager, the processing of outgoing connections is handled by the MSCP. The selection of a line for switched controller descriptions, the selection of a network interface description for switched lines, call-out processing, and error processing or notification for network interface description failures are all responsibilities of the MSCP.

Answer Manager

The answer manager is the "clearinghouse" for all incoming connections over ISDN, and the answer manager implements the answer management function described earlier. The ISDN informs the system of incoming calls by sending a SETUP message on the D-channel, which contains information such as the number called, the number calling, and the type of information and protocol that will be used for the connection. This SETUP message is sent to the answer manager by the ISDN I/O manager to determine if the call should be accepted by the system. Based on the information in the SETUP message, the answer manager must find a line description that is in the proper state and properly configured to accept the call. If a suitable match is found, it instructs the ISDN I/O manager to accept the call.

The answer manager is a separate, high priority function to help ensure that the system can respond rapidly to incoming calls. Most ISDN networks allow only limited time for a call to be accepted, so the answer manager must be able to process incoming calls rapidly, something that it might not be able to do if it were not a dedicated function on the system.

ISDN I/O Manager

The ISDN I/O manager provides the system with a single interface to the call control functions on the D-channel of an ISDN interface. The management services control point starts the ISDN I/O manager when the ISDN network interface description is varied on. The ISDN I/O manager controls the activation of the ISDN interface and the startup of the D-channel protocol stack so that the system can begin to make and receive ISDN calls. The ISDN I/O manager also manages all incoming and outgoing calls over the ISDN interface.

The management services control point requests outgoing connections over the ISDN interface by sending a message to the ISDN I/O manager. The ISDN I/O manager then instructs the ISDN call control layer (Q.931) in the I/O processor to make the call, and it informs the MSCP when the ISDN connection is established.

Incoming connections from the ISDN network are received by the ISDN call control layer, which then signals the ISDN I/O manager. The ISDN I/O manager then informs the answer manager, which determines if the call can be accepted. If the call is accepted by the answer manager, the ISDN I/O manager instructs the ISDN call control layer to accept the call, and informs the MSCP when the ISDN connection is established.

IDLC Line I/O Manager

The ISDN data link control (IDLC) line I/O manager supports the new ISDN data link control over an ISDN B-channel. IDLC is a duplex protocol similar to SDLC but tailored to take advantage of the performance benefits of ISDN. The IDLC line I/O manager provides a transparent interface to the ISDN network for the existing SNA station I/O managers.

The management services control point starts the IDLC line I/O manager when an IDLC line description attached to a permanent ISDN B-channel is varied on, or when a switched connection over an ISDN B-channel is connected to an IDLC line description. The line I/O manager manages the activation of the data link control and the data interface provided to the station I/O managers through its interface to the I/O processor ISDN data link control (Q.922).

Conclusion

The AS/400 system's Version 2 implementation of ISDN meets each of its design objectives.

Integration is achieved by incorporating the advanced functions of ISDN into the system's object and communications task structure. This integration provides for maximum performance and results in a more intuitive view of ISDN for the user.

Flexibility is provided with communications objects that are easily configured. They describe the call-related information and the user's ISDN subscription. By providing defaults for many network specific parameters, the user may tailor his configuration to the network's requirements.

ISDN's complexity is simplified by reducing most parameters into a set of system-provided defaults, chosen by the type of network, and by allowing the user to avoid most of the call-related information with the system default connection list.

Future enhancements and refinements are possible due to an object-based task structure that can expand to represent and employ many of the connectivities and advanced functions of ISDN.

Transmission Control Protocol/Internet Protocol

Describes TCP/IP protocol and the AS/400 implementation of TCP/IP.

Robert L. Dick, Jonathan P. Beierle, and
Mark L. Bauman

Introduction

The growing telecommunications industry has created a requirement for interoperability between dissimilar vendor equipment. One of the ways this requirement can be fulfilled is through the use of Transmission Control Protocol/Internet Protocol (TCP/IP). **TCP/IP** is a collection of vendor-independent communications protocols that support peer-to-peer connectivity functions for both local and wide area networks. TCP/IP is used to organize computers and communications equipment into a network. This network can be a single local area network or a set of networks.

The use of TCP/IP is pervasive in the current telecommunications marketplace. An example of this marketplace dominance is the international collection of networks known as the **Internet**. The number of networks in the Internet is currently in the thousands. TCP/IP ties all of the Internet's individual networks together so that any individual user can reach systems anywhere else in the Internet. TCP/IP allows all nodes in the Internet to communicate as if they were on the same physical network, regardless of their specific hardware or software architecture, and in this way TCP/IP provides interoperability.

TCP/IP is an international standardized protocol. TCP/IP and its constituent protocols are standardized by the Internet Activities Board. The document that provides the standard for a particular protocol is called a Request For Comment (RFC). There are over 1,000 RFCs available today. RFCs are available from the Data Defense Network (DDN) Information Center at Government Systems, Inc. (Network Information Center) in Chantilly, Virginia.

TCP/IP Structure

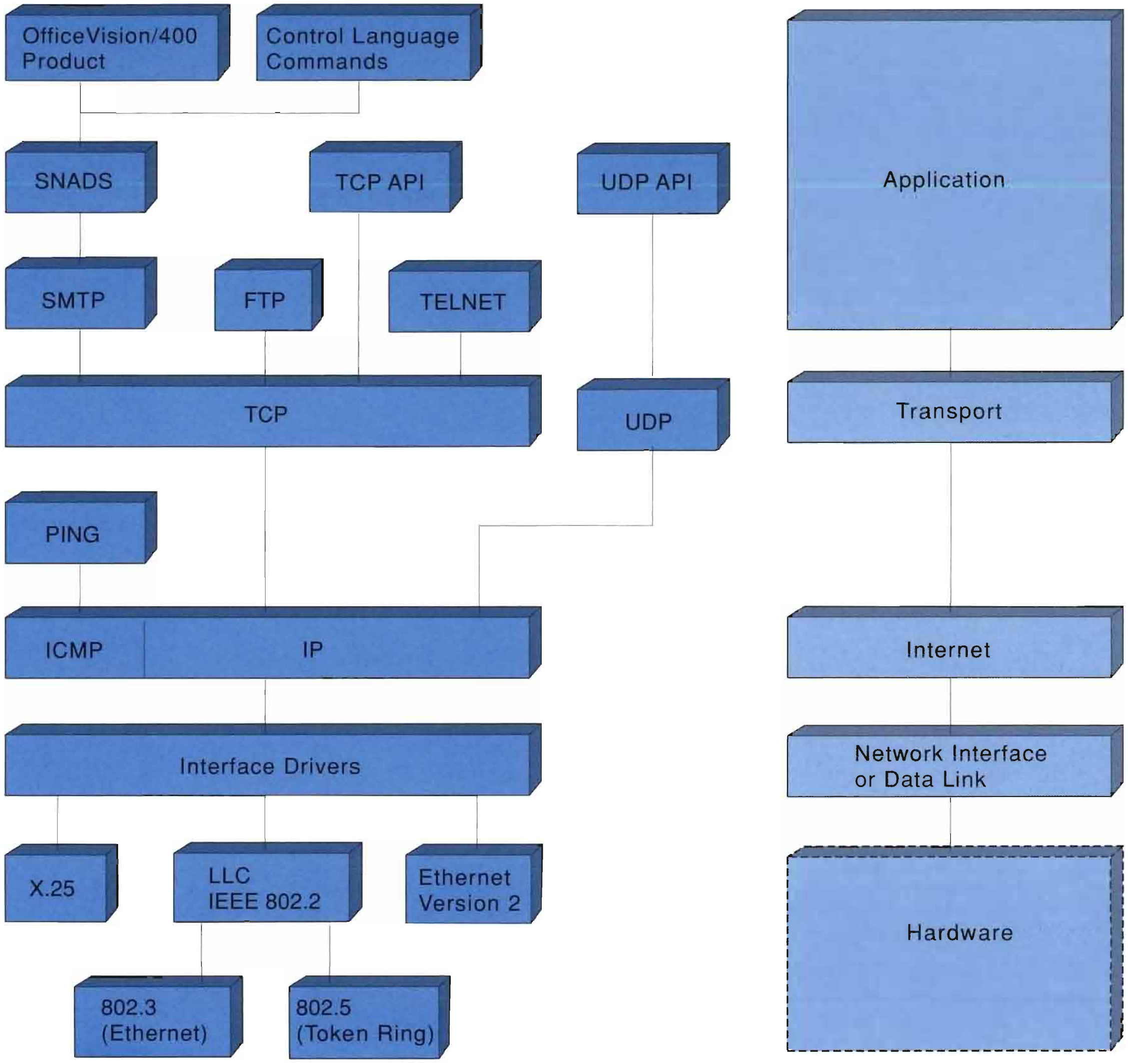
TCP/IP consists of a four-layered structure of protocols (see Figure 54 on page 103) ranging from low-level, hardware-dependent programs to high-level applications. Each TCP/IP layer provides services to the layer above it and uses the services provided by the layer below it.

Data Flow

In the computer networks that use TCP/IP protocols, information is transmitted between nodes in the form of packets. A packet includes an Internet Protocol (IP) header and data. A packet can be a complete IP datagram or a fragment of an IP datagram. (A **datagram** is the basic unit of information in the TCP/IP protocol, consisting of a source address, destination address, and data.) After packets are received by the network interface or data link layer, they are passed to the internet layer. The internet layer is implemented as the IP. Datagrams are reassembled if necessary, and stripped of the IP header before being passed to the next higher protocol layer called the transport layer. The transport layer is usually implemented as Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). The transport layer takes the datagrams (now called segments), strips off the transport layer's header, reassembles the data stream, and passes this data to the application layer. The application layer then processes the data.

Hardware Layer

The hardware layer consists of the hardware-specific network protocols. These protocols include token ring, Ethernet, X.25, and others. The network protocols are not really a part of the TCP/IP protocol. However, the IP must directly interface to these network protocols to operate across a network.



API	=	Application Program Interface
FTP	=	File Transfer Protocol
ICMP	=	Internet Control Message Protocol
IEEE	=	The Institute of Electrical and Electronics Engineers
LLC	=	Logical Link Control
PING	=	Packet InterNet Groper
SMTP	=	Simple Mail Transfer Protocol
SNADS	=	Systems Network Architecture Distribution Services

TECH119-3

Figure 54. *Relationship of AS/400 TCP/IP Implementation to TCP/IP Layers*

Network Interface or Data Link Layer

The network interface or data link layer provides the bridge between the IP and the data link and hardware layer. These are commonly referred to as drivers.

Internet Layer

The IP provides the basic transportation rules for communications between **hosts** (unique internet addresses with associated system names) on the different networks that make up an internet (an individual network that may or may not be part of the Internet). IP is responsible for implementing the internet concept, which is accomplished by routing datagrams from a host on one network through a series of gateways to a host on another network. IP communicates directly with the network interface or data link layer. IP makes the connected networks appear to the layers above as a single, virtual network. At the internet level, all addressing is host to host, using fixed length addresses to identify source and destination hosts. The protocol layers above only need to know each host's internet address to make a connection. Because IP does not acknowledge receiving a datagram and is not responsible for retransmitting or providing flow and error control, reliable delivery must be ensured by a higher level protocol, such as TCP, or an application.

Internet Control Message Protocol (ICMP) is an integral part of the IP's implementation. The ICMP provides for error and control messages between **host systems** (peer computers in a network) and gateways. Gateways and host systems use ICMP to send reports of problems. ICMP also includes an echo request or reply to test whether a destination can be reached and is responding. This is commonly referred to as PING (Packet InterNet Groper).

Transport Layer

The Transmission Control Protocol (TCP) provides a reliable delivery of a stream of bytes in sequence. It is a connection-oriented transport mechanism. TCP takes a stream of data, breaks it into segments (a TCP header and application data), sends each one individually (using IP), and then reassembles the segments back into the original stream on the receiving node. If any segments are lost or damaged during transmission, TCP detects this fact and resends the missing segments. TCP communicates without a regular or predictable pattern (asynchronously) with applications and assumes that IP is the underlying protocol.

The **User Datagram Protocol** (UDP) provides an application-to-application delivery service with a minimum of protocol overhead. Unlike TCP, UDP is connectionless and does not offer guaranteed delivery of data. An application should use UDP to avoid the TCP overhead of connecting and disconnecting; the application takes responsibility for acknowledgment and retransmission to ensure reliable data transfer.

Application Layer

The application layer consists of several independent protocols that implement commonly used applications. Most TCP/IP implementations offer these three common application protocols:

- **TELNET Protocol (TELNET)** allows remote logon to hosts within a TCP/IP network.
- **File Transfer Protocol (FTP)** allows copies of files to be exchanged between hosts running TCP/IP.
- **Simple Mail Transfer Protocol (SMTP)** allows electronic mail to be exchanged and manipulated between hosts running TCP/IP.

Almost all TCP/IP applications incorporate the idea of a server and a client. A **server** is a designated computer or application on the network that makes specialized services available to other computers on the network (the clients) and handles requests from programs on those computers. For example, a TELNET server provides TELNET application services to any TELNET client that requests them. The **client** can be any program that communicates with or uses the services of TCP/IP. Different computers provide specialized services for the benefit of the entire network.

Host Identification

Each node on a network is known as a host and has a unique address called an internet address. This address is a 32-bit integer. An internet address is expressed in the form www.xxx.yyy.zzz, where each field is the decimal representation of 1 byte of the address. For example, the address whose hexadecimal representation is `X'82638001'` is expressed as `130.99.128.1`.

Within private networks, administrators can assign addresses according to their preference. However, if the network connects to the Internet (the collection of government, military, research, and university networks), then the internet addresses must be assigned by Government Systems, Inc. (Network Information Center) in Chantilly, Virginia. This prevents any two hosts on the Internet from using a common internet address.

TCP/IP uses internet addresses to route datagrams between systems in a network. Because internet addresses can sometimes be difficult to remember, another naming convention (the domain name) provides an easier way to identify systems in a network.

A **domain name** identifies the system within a hierarchy of systems and can be used by remote servers to associate an internet address with the domain name of the system. Domain names consist of labels that are separated by periods (for example, ABC.DEF.XYZ). A shortened version of a domain name (a **host name**) can also be used to identify a system and the associated internet address. Host names are a sequence of characters.

It is a common practice to use hierarchical names that allow predictable extensions of a network for change and growth. Domain names normally reflect the delegation of authority or a hierarchy used to assign them. For example, the name *SYS1.MFG.ABC.COM* can be broken down into the following:

COM: All commercial networks.

ABC.COM: All systems in the ABC Company's commercial network.

MFG.ABC.COM: All manufacturing systems in the ABC Company's commercial network.

SYS1.MFG.ABC.COM: A host named *SYS1* in the manufacturing area of the ABC Company's commercial network.

The COM designation is one of several domain names used when connecting to the Internet. Some of the domain names follow:

COM	Commercial organizations
EDU	Educational institutions
GOV	Government institutions
MIL	Military groups
NET	Major network support centers
ORG	Organizations other than those listed above
ARPA	Temporary ARPANET domain
Country code	Countries other than USA

TCP/IP on the AS/400 System

The AS/400 system implements TCP/IP as a licensed program called TCP/IP Connectivity Utilities/400. This licensed program provides another way for the AS/400 system to operate in telecommunications networks.

TCP/IP has been available as a product on the AS/400 system since Version 1 Release 2. Currently, it supports IP, ICMP, Address Resolution Protocol (ARP), TCP, UDP, FTP, SMTP, TELNET, Network Status (NETSTAT), TCP and UDP programming interface, and extensive configuration support.

The AS/400 TCP/IP product runs in a subsystem called QTCP. The QTCP subsystem must be started before any TCP/IP application, such as TELNET, FTP, SMTP, PING, or NETSTAT, can be used.

A number of CL commands, such as NETSTAT, TELNET, FTP, Add TCP Link (ADDTCPLNK), Work with Names for SMTP (WRKNAMSMTP) and others, are part of the TCP/IP product on the AS/400 system. The CL commands are used to perform TCP/IP functions and configuration.

The following topics detail the major application protocols and interfaces implemented in the TCP/IP Connectivity Utilities/400 licensed program as of Version 2 Release 1.

TELNET Protocol

The TELNET protocol allows a system (the TELNET client) to access and use the resources of a remote system (the TELNET server) as if the TELNET client's work station were locally connected to the remote system.

TELNET is implemented in two parts on the AS/400 system: the TELNET client and the TELNET server.

The TELNET protocol provides a mechanism for the client and server to negotiate options that control the operating characteristics of a TELNET connection. Part of these negotiations involve determining the optimal terminal type that is supported by both the client and server. Depending on the terminal type negotiated, the TELNET server or TELNET client operates in one of three possible operating modes on the AS/400 system: ASCII line mode, 5250 full-screen mode, or 3270 full-screen mode. The functions available in a TELNET session depend on the operating mode.

TELNET Server

The TELNET server is automatically set up to support TELNET connections when the QTCP subsystem is brought up.

When a TELNET session is initiated, a virtual device is allocated by the TELNET server. A virtual device description is associated with each TELNET session on the TELNET server system. A **virtual device** is a device description that does not have hardware associated with it. It is used to form a connection between a user and a physical display station attached to a remote system. It provides information about the physical display station to the programs on the remote system, just as the physical device description on the system provides information about the physical display station to the programs on the local system.

When a TELNET connection to an AS/400 system is made in 5250 or 3270 full-screen mode, the TELNET server attempts to match a virtual device description with a device type and model similar to the device on the client system. In 5250 or 3270 full-screen mode, the AS/400 system automatically allocates (and creates, if necessary) a virtual device. It then sends the AS/400 sign-on display when the TELNET connection is established.

Other TCP/IP hosts may not support either 3270 or 5250 full-screen modes. Lack of this support may mean the TELNET connection becomes a line-by-line interaction. This is known as ASCII line mode.

If ASCII line mode is negotiated, the AS/400 system does not automatically send a sign-on display. For ASCII line mode, a customer application that displays data must be active. Also, a network virtual terminal (NVT) controller description and associated NVT device descriptions must be created before the TELNET connection is attempted. ASCII line mode is implemented this way because the AS/400 system operates in full-screen mode and has screens with many input fields. It is difficult, and most likely unusable, to map these screens to a line-mode device.

TELNET Client

The TELNET client is automatically set up to support TELNET connections when the QTCP subsystem is brought up.

If 3270 or 5250 full-screen mode is negotiated, the keyboard mapping from the AS/400 system to the remote system must be considered. The keyboard mapping used depends on the type of physical display station and its attached keyboard, the type of display station negotiated with the TELNET server, and how the physical display station is attached to the AS/400 system, such as twinaxial, PC Support, or 3270 remote attachment.

If ASCII line mode is negotiated, the AS/400 TELNET client application maps the incoming data to EBCDIC and emulates a line-mode work station. The data received from the remote host system is displayed on the scrollable output area of the screen. All outgoing data is encoded in 7-bit ASCII format. The data is not formatted; however, it does contain carriage returns, line feeds, and spaces.

TELNET Control Functions

As part of most TELNET implementations, there are functions referred to as control functions. These control functions control work station processing on the server system when in a TELNET session. These functions include:

- Interrupt process (IP): Interrupts, cancels, or suspends an operation that has started on the server system.
- Query connection status or Are You There (AYT): Provides a message or an alarm to indicate that the server system is still running.
- Discard remote output data or Abort Output (AO): Allows a process that is generating output to run to completion (or to reach the same stopping point it would reach if running to completion) without sending the output to the work station.
- Clear the data path or SYNCH: Discards all characters (except TELNET commands) between the client and the server systems.
- End TELNET session or QUIT: Ends the TELNET session and closes the TCP/IP connection to the server system.

File Transfer Protocol

The File Transfer Protocol (FTP) allows users to send or receive copies of files to or from remote systems across a TCP/IP network. In addition, FTP provides functions for renaming, appending, and deleting files.

FTP supports the following functions:

- Sending or receiving physical files, source files, and logical files.
- Transferring binary files "as is."
- Sending text files in EBCDIC format or translating them to ASCII (the default format).

- Creating libraries, files, and members with AS/400 FTP server subcommands.
- Submitting and accepting remote CL commands.
- Running FTP unattended (in batch). FTP can be run unattended using either a REstructured eXtended eXecutor (REXX) language procedure or a CL program. An example REXX procedure and an example physical file member are shipped as part of the TCP/IP product.
- Using ASCII and EBCDIC mapping tables for AS/400 FTP servers and FTP clients to map incoming and outgoing data.

AS/400 FTP currently does not support the following:

- Sending or receiving save files
- Transferring individual records within files
- Transferring files greater than 16 million bytes
- Automatically sequencing a source physical file member after its transfer to the AS/400 system

Like TELNET, FTP supports both client and server functions. To start an FTP client session with another TCP/IP host, the FTP or the Start TCP FTP (STRTCPFTP) CL command may be used. When the FTP connection is established to the remote TCP/IP host, the FTP client session is prompted for a user ID and a password. This maintains the security in the file transfers between hosts.

The FTP server function is enabled when the QTCP subsystem is started.

Within the FTP connection, commands are used to transfer files or change the FTP connection characteristics. These commands are called FTP subcommands. Some of the FTP subcommands accept a file name as a parameter. The file naming conventions of the system on which the file will reside must be used. The format used to name a file is dependent on the system. Some systems limit the length of a file name, and some systems are uppercase and lowercase sensitive. For example, the AS/400 system uses the format:

```
Libraryname/Filename.Membername
```

and UNIX**-based systems allow specifications such as:

```
/etc/hosts
/usr/jack/test.c
/usr/*
```

Files may be transferred individually or in multiples. Also, multiple files that include a common subpart in their names can be transferred by using wildcard characters.

The MODE subcommand specifies how the bits of data are transmitted, and the TYPE subcommand defines the way in which the data is represented.

The appropriate transmission attributes must be used to transfer files between two systems to preserve the content and structure of the files. Table 1 shows how to set these attributes for different systems. An EBCDIC text file normally contains standard, displayable characters. An ASCII text file normally contains standard, displayable characters and end-of-record characters. A binary file can contain any character and is transferred "as is."

Table 1. Recommended Methods for Data Transfer

Transfer between System Types	Data Type	Transfer Type, Mode
AS/400-to-AS/400	Text	EBCDIC, Stream
AS/400-to-ASCII	Text	ASCII, Stream
ASCII-to-AS/400	Text	ASCII, Stream
ASCII-to-AS/400-to-ASCII	All data	Binary, Stream

When a connection is established with the remote system, FTP subcommands can be entered. Each subcommand has a unique purpose. The AS/400 FTP *client* supports the following subcommands (the purpose is apparent by the name of the command in most cases):

ACCT APPEND ASCII BINARY
CD CLOSE DELETE DIR
EBCDIC GET HELP LOCSTAT
LS MDELETE MGET MODE
MPUT NOOP OPEN PASS
PUT PWD QUIT QUOTE
RENAME SENDPORT SENDSITE SITE
STATUS STRUCT SUNIQUE SYSCMD
SYSTEM TYPE USER

The AS/400 FTP *server* supports the following subcommands:

ABOR ADDM ADDV APPE CRTL CRTP
CRTS CWD DBUG DELE DLTF DLTL
HELP LIST MODE NLST NOOP PASS
PASV PORT PWD QUIT RCMD RETR
RNFR RNTO SITE STAT STOR STOU
STRU SYST TIME TYPE

FTP architecture limits FTP server subcommand names to 4 characters or less. Thus, the specialized AS/400 FTP server subcommands of ADDM, ADDV, CRTL, CRTP, CRTS, and DLTL are really abbreviated names of equivalent (but longer) AS/400 CL commands. When the AS/400 remote server receives these subcommands, it passes them to the AS/400 subcommand interpreter as follows:

- ADDM = ADDPFM (Add Physical File Member)
- ADDV = ADDPVLM (Add Physical File Variable Length Member)
- CRTL = CRTLIB (Create Library)
- CRTP = CRTPF (Create Physical File)
- CRTS = CRTSRCPF (Create Source Physical File)
- DLTF = DLTF (Delete File)
- DLTL = DLTLIB (Delete Library)

A unique FTP server subcommand supported by the AS/400 system is the remote command (RCMD) subcommand. It runs CL commands on the FTP server system. The length of the RCMD subcommand string is limited to 94 characters. Because no prompting is available for the RCMD subcommand, the RCMD subcommand string must include all necessary parameters to run the CL command.

The following is an example of submitting a job from a remote system using the FTP server subcommand of RCMD:

```
QUOTE RCMD SBMJOB JOB(FTPS) JOBD(QTCP/QTCPFTPS) RTGDTA(*JOBD)
   RQSDTA(*JOBD)
```

Simple Mail Transfer Protocol

The Simple Mail Transfer Protocol (SMTP) is used to send or receive electronic mail. For consistency with other AS/400 mail functions, SMTP is coupled with Systems Network Architecture (SNA) distribution services (SNADS). SNADS is part of the Operating System/400 licensed program, and it contains extensions to support SMTP. Because SMTP is coupled to SNADS, users can send mail to various types of users (not just SMTP users) with one consistent user interface. The distribution services (receive, forward, and send electronic mail) for SAA OfficeVision/400* functions are provided by SNADS.

SMTP supports the following functions:

- OfficeVision/400 notes and messages
- AS/400 documents in final-form text (FFT) format
- Documents and messages through SNADS using IBM Displaywriter or Personal Services/PC
- Documents and messages using AS/400 commands

  Note: If the SAA OfficeVision/400 product is not installed, users can still send and receive SMTP messages without using OfficeVision/400.

- SMTP notes as an intermediate TCP/IP hop on an SMTP distribution
- Transporting binary files across TCP/IP links

SMTP does not support the following:

- Distributing objects
- Transferring revisable-form text (RFT) documents

  Note: If a document is saved on the AS/400 system in RFT format, it must have its format changed to FFT before sending the document.

- Sending information using the Send Network Message (SNDNETMSG), Send Network Spooled File (SNDNETSPLF), or Send Network File (SNDNETF) command
- Transferring documents exceeding 16 million bytes

Like TELNET and FTP, SMTP supports both client and server functions on the AS/400 system.

One major advantage of the AS/400 SMTP implementation is that a network that uses SNADS and an office application to send and receive mail can connect to another network that uses TCP/IP and SMTP for mail distribution. This can be accomplished by using an AS/400 system that is configured with TCP/IP and SNADS. This connection is called a gateway. See Figure 55 on page 109.

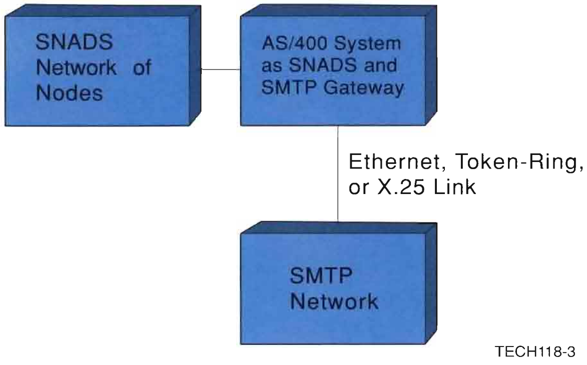

SNADS Network of Nodes
AS/400 System as SNADS and SMTP Gateway
Ethernet, Token-Ring, or X.25 Link
SMTP Network
TECH118-3

Figure 55. *Gateway between a SNADS and an SMTP Network*

When a distribution is sent from a SNADS network to an SMTP network, the systems in the SNADS network use SNADS to send mail to the gateway. At the gateway, the SNADS distribution is converted to an SMTP distribution, which is then sent to the final destination using the SMTP function. Similarly, when a distribution is sent from an SMTP network to a SNADS network, the SMTP function is used to send mail to the gateway. At the gateway, the SMTP distribution is converted to a SNADS distribution, which is then sent to the final destination.

When an AS/400 system is used as a gateway, it must be configured with TCP/IP and SNADS; however, other systems in the SNADS network need not have TCP/IP installed and need no special configuration.

TCP/UDP Programming Interface

An enterprise often has unique interoperability requirements for its private networks. This means that the enterprise must provide its own applications to fulfill these unique requirements. One way this is accomplished is with the AS/400 TCP/UDP programming interface. The TCP/UDP programming interface provides a system programmer with a programming interface to TCP or UDP as a set of procedure calls from an AS/400 Pascal program.

Network Management

When an AS/400 system is running TCP/IP and users are using FTP, TELNET, and SMTP, a system administrator needs to monitor and control the network status of the AS/400 system. The network status (NETSTAT) function on the AS/400 system provides the information about the status of TCP/IP network routes, links, and connections on a local system. This function also allows the administrator to end TCP/IP connections and to start and end TCP/IP links.

There are three options available for the NETSTAT function on the AS/400 system:

- Work with TCP/IP Link Status

  The link status display provides the most current summary of link activity. This display provides options to start and end TCP/IP links.

- Display TCP/IP route information

  The route information function allows the system administrator to view information about TCP/IP routes that are currently configured on the system.

- Work with TCP/IP Connection Status

  The connection status display provides options to display and end TCP/IP connections between local systems and remote systems.

Conclusion

Customers often have equipment from more than one vendor to solve their particular data processing needs. TCP/IP is a suite of protocols that allow equipment from many vendors to interoperate all the way from the low-level physical interfaces up through the actual data exchange at the application level. TCP/IP Connectivity Utilities/400 is a licensed program that services customer needs through the implementation of the TCP/IP suite of protocols.

Acknowledgments

The authors would like to acknowledge the contributions of Paula K. Muth, Rob V. Downer, and Paul R. Chmielewski in supplying information used in the creation of this article.

Systems Management

Describes the many advancements in systems management functions of the AS/400 system.

Earl W. Emerick and James R. Morcomb

Introduction

Systems management and the electronic customer support functions of the AS/400 system bring customers a set of capabilities that allow them to be more productive in both a stand-alone system and in a network environment. The combination of these two key areas are cumulatively referred to as **systems management**.

Systems management functions are ever growing in breadth and depth with a focus on increased productivity across a broad range of users, network types, and systems. This article discusses the background and foundation provided in Version 1 and the subsequent enhancements in Version 2 of the AS/400 system and systems management solutions targeted at helping users manage their systems.

Background

The AS/400 system is being installed in environments of ever-increasing complexity. Just a few years ago, mid-range computers were primarily installed as small stand-alone systems in small peer networks and as nodes on System/370* host networks. However, the AS/400 system is now being installed in larger numbers in complex networks since its initial availability. In many cases a System/370 or a System/390* host manages these networks using NetView* and INFO licensed programs. In addition, more and more complex networks are now hosted and managed through the AS/400 system. In both cases the system is expected to manage the attached networks (wide area network [WAN] or local area network [LAN]) and downstream systems, such as Personal System/2 (PS/2*) and RISC System/6000* (RS/6000) systems.

In Version 1 Release 1, the AS/400 system established a new benchmark of excellence in service support and systems management through its electronic customer support capability. In Releases 2 and 3 of Version 1, the electronic customer support and systems management capability were enhanced with new operating system functions and licensed programs, such as the AS/400 Systems Management Utilities.

The focus of Version 1 was to build a foundation for systems management function on the AS/400 system and to establish an initial capability for central enterprise management using the system. Much of the foundation has been built and the introduction of Systems Management Utilities provided a major stepping stone into the central enterprise management environment.

The introduction of SystemView* technology in September 1990 and the announcement of Version 2 Release 1 in April 1991 also establish new directions for systems management on the AS/400 system.

To better understand the technical and functional growth in systems management, a summary of the Version 1 major capabilities follows.

Version 1 Foundation

In Version 1, the architectural foundation and functional building blocks for AS/400 systems management were established. Figure 56 on page 111 provides an overview of the key functional building blocks.

For more information about these building blocks, refer to the article “Electronic Customer Support” in the *IBM Application System/400 Technology*, SA21-9540.

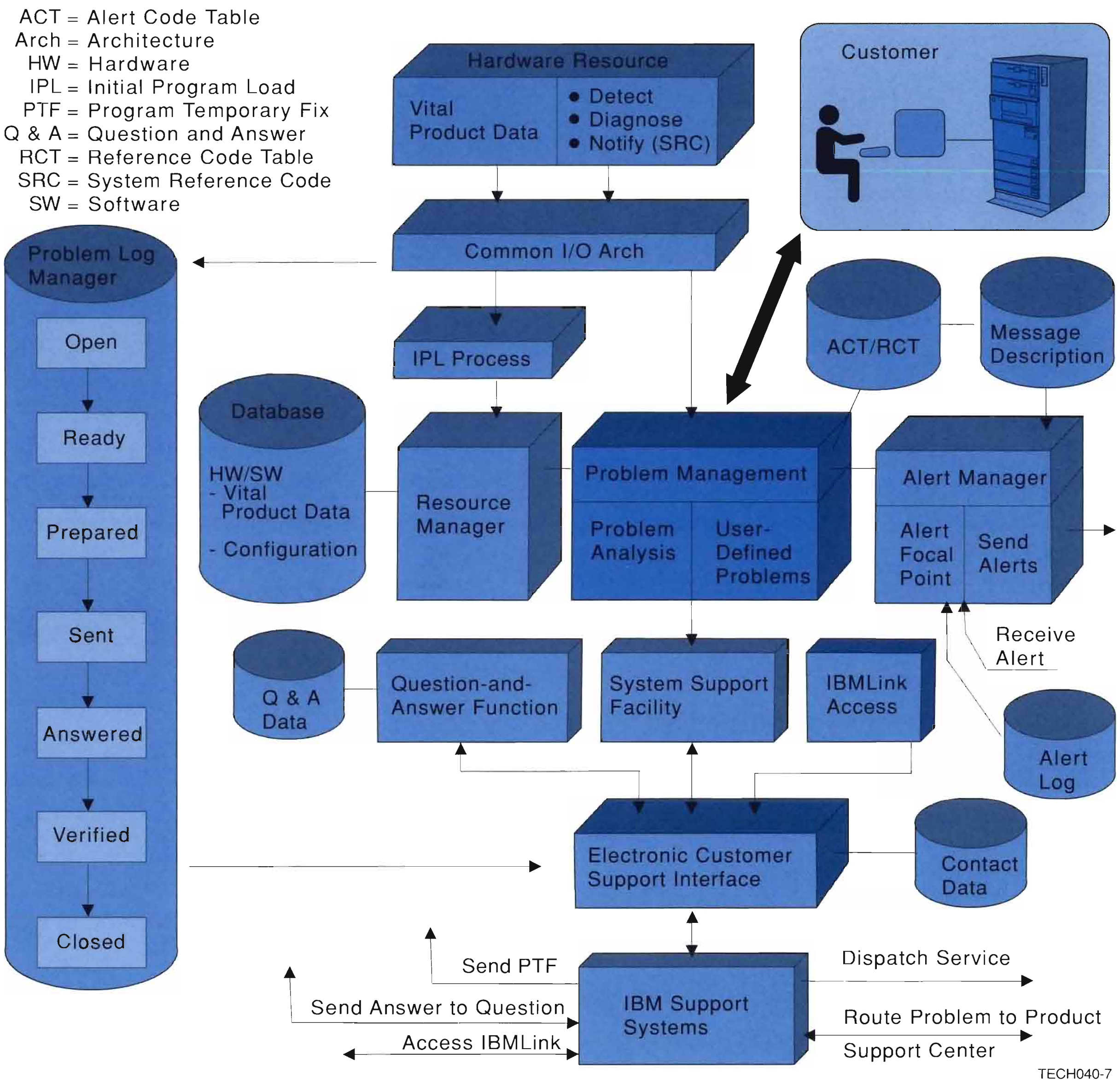


Figure 56. *Building Blocks*

Fundamental to the systems management solutions are the architectures and design concepts, which are integrated into all levels of the AS/400 system.

AS/400 systems management is built on strategic architectural foundations. Key architectures are:

- Common input/output (I/O): This architecture provides the basis for system management communications between bus-attached I/O devices and the Operating System/400 (OS/400) licensed program.
- Vital product data (VPD): VPD provides the common definition of the product information embedded in all levels of hardware and software on the AS/400 system.
- Systems Network Architecture (SNA) alert: The OS/400 alert manager provides a complete implementation of the IBM SNA alert architecture.
- System reference codes (SRC): SRCs provide a standardized approach for communicating local system problems to OS/400 programs.
- Advanced peer-to-peer networking (APPN): The APPN architecture is implemented in the operating system and provides the basis for network connectivity, local and remote transparency for centralized system management solutions, and network management for the AS/400 system.

Two basic design concepts are essential to enable the integrated and automated systems management capability of the system. They are:

- Self-identification: The hardware and software components of the AS/400 system contain self-identifying information called VPD. VPD is essential to the automated processes for automatic configuration, resource management, problem management, change management, system upgrade, and systems management solutions.
- First failure data capture (FFDC): Problem management on the AS/400 system is based on the FFDC capability. FFDC is built into all levels and components of the system. Problems are detected at the point of origin, and the necessary data to resolve the problem is captured. The data is automatically analyzed to identify the recommended corrective action. The operating system is notified of the problem, and supporting data is logged to enable follow up if the recommended corrective action does not resolve the problem. One major design point in this process is to minimize problem re-creation.

Three additional design concepts provide a strategic advantage for the implementation of systems management on the AS/400 system. system. They are:

- Integration: AS/400 systems management solutions are highly integrated. Function and data are shared between OS/400 systems management components and systems management applications. In addition, the systems management functions use other integrated system functions, such as the user interface manager, storage and data management, security management, and communications support.
- Automation: A basic concept of AS/400 systems management is to simplify or eliminate user tasks through automation. This was accomplished with integrated process managers for functions such as software installation, automatic configuration, PTF management, scheduled performance data collection, first failure data capture, remote unattended operations, and transparent connectivity to support systems using electronic customer support functions.
- Electronic customer support: This provides the components, integrated in the OS/400 licensed program, that help automate service and support for all levels of hardware and software on the AS/400 system.

Version 1 Solutions

In Version 1, solutions were provided in all disciplines of systems management. The essential functions to the overall systems management capability on the AS/400 system included:

- Alert manager and alert log
- Problem manager and log
- Problem analysis utilities and error or event logs
  - Problem analysis and resolution
  - User-identified problem resolution
- Electronic customer support
  - Hardware and software service request
  - PTF ordering and delivery
  - Question-and-answer function
  - Access to IBMLink
  - IBMLink file transfer
- Automatic configuration
- Resource manager and database
- Copy screen image
- Display station pass-through
- Performance monitor and data logging
- Change management and PTF distribution and installation
- Remote unattended operations support
  - Remote power control
  - Remote IPL

In addition, two licensed programs provided significant additional function for managing the AS/400 system in Version 1.

- IBM AS/400 Performance Tools: This program provides a set of tools to help the AS/400 customer monitor, track, diagnose, and model system performance. The tools provide interactive graphs and text-based views of the performance data collected by the OS/400 performance monitor. Functions in AS/400 Performance Tools include the ability to:

- Produce reports depicting resource and system wide use
- Select groups of users and resources to report on
- Display graphical and character reports of collected data
- Display real-time system activity
- Do capacity planning based on collected data and modeled growth scenarios

- IBM AS/400 Systems Management Utilities: This program provides the capability for an AS/400 system to become a manager for all of the AS/400 agent systems in a network (see Figure 57). Using Systems Management Utilities, the managing system can:

  - Receive and process PTF orders from any AS/400 agent system
  - Answer problems from AS/400 agent systems
  - Perform remote problem analysis
  - Track all problems centrally and maintain a central problem log
  - Provide centralized electronic customer support for all AS/400 systems in the network

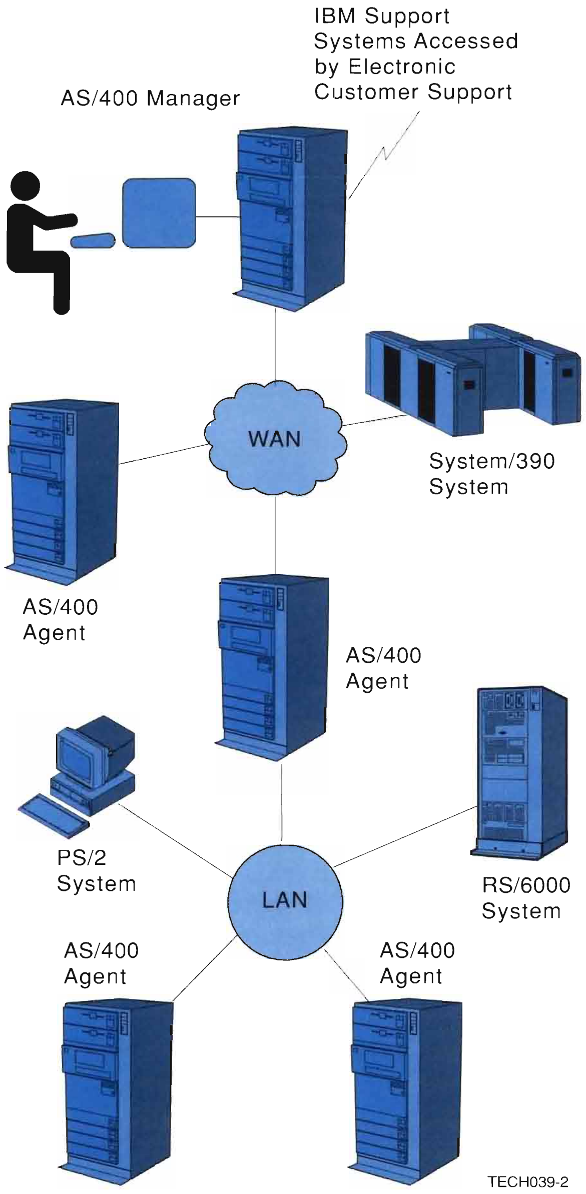


Figure 57. *AS/400 Manager and Agent Systems in a Network*

The agent systems can:

- Send PTF orders to a managing system
- Request service from the AS/400 manager system
- Have problems remotely analyzed by the manager system
- Track all problems relative to their problem domain

Version 1 of the AS/400 system successfully met three key goals:

- Support initial customer needs for managing the AS/400 system in a single-system environment.
- Provide initial functions for centrally managing multiple AS/400 systems and the network that connects them.
- Build a strategic state-of-the-art foundation for a full systems management solution on the system.

Refer to Figure 58 on page 114 for an overview of the systems management structure and each layer's design objectives. Integration, automation and well-defined component responsibilities are key to all levels of the structure.

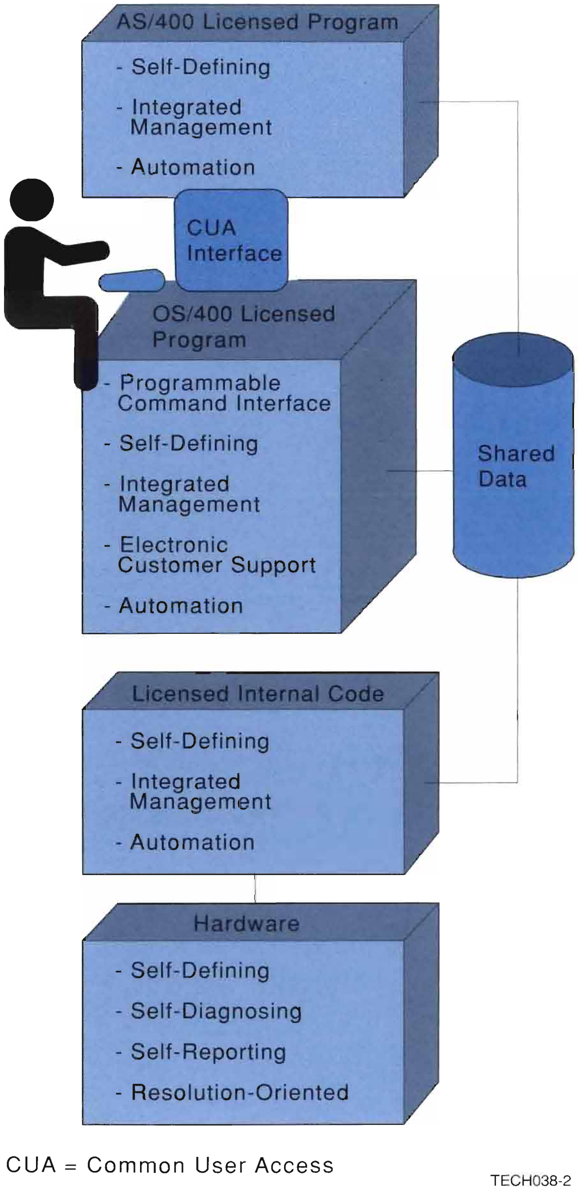


Figure 58. *Structure Overview*

Version 2

Since Version 1, many new additions have been made to the systems management direction and functions. One significant area is the SystemView additions and direction. The remainder of this article discusses functional additions and a portion of SystemView technology. For more information on SystemView technology, refer to the following publications:

- *SystemView Concepts*, SC23-0578
- *SystemView End-Use Dimension Consistency Guide*, SC33-6472
- *SystemView and the Application System/400 System*, GA21-9607

SystemView Technology Introduction

SystemView technology was introduced in September 1990 as the IBM structure for systems management. The SystemView structure provides a model, which the AS/400 system will implement over time, for building consistent, heterogeneous systems management solutions across all Systems Application Architecture (SAA) platforms. The structure is made up of three dimensions (refer to Figure 59 on page 115). Note the similarity between these dimensions and the Common User Access (CUA) interface, shared data, and central system structure in Figure 58.

The **end-use dimension** provides the integrated work place for SystemView applications. The SystemView work place is iconic, object oriented, task related, and CUA compliant.

The **data dimension** provides a framework for common data definition and data sharing between SystemView applications. The data model is object-oriented with an SAA SQL interface.

The SystemView **application dimension** is divided into six disciplines (business, change, configuration, operations, performance, and problem management). The application function is integrated within and across disciplines to provide overall sharing of a function. Examples of function sharing in the AS/400 system would be problem management's use of the change management function to resolve a software problem or change management's use of resource data (VPD) to determine the product level.

The AS/400 system is positioned as a strategic SystemView managing system. The architecture, design concepts, and functional components of its systems management solutions place the system in a good position for implementing the new SystemView structure. The following topics discuss key technologies in terms of each of the three dimensions of SystemView technology.

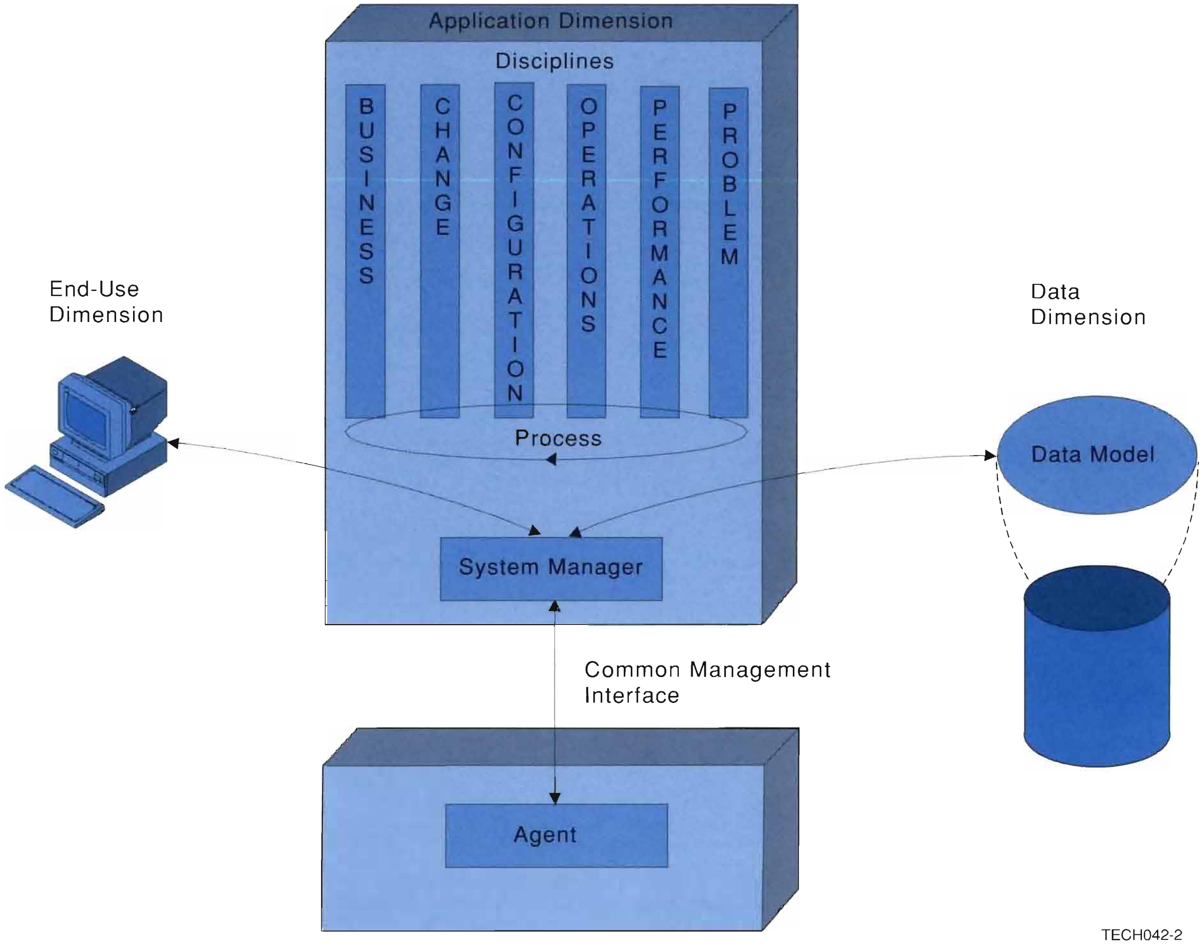


Figure 59. *SystemView Dimensions*

End-Use Aspects of Systems Management

The AS/400 system spans a large range of users from a single-user environment, a departmental node in a System/390 network, and as the host of a large network of complex systems (both personal computers and AS/400 systems).

The user interface is consistent across the range of user environments on the AS/400 system. It provides guided access for new users as well as fast path access for experienced users to a set of systems management functions.

All of the AS/400 systems management applications supplied by IBM use the same user interface. This provides a common "look and feel" to the systems management functions and should ease users' learning of new functions as new functions are provided.

The systems management user interface is supported by integrated process managers that provide automated processing of underlying functions, greatly simplifying the overall user task that is to be performed. In Version 2 Release 1 of the AS/400 system, the new licensed program IBM SAA SystemView System Manager/400 was announced. System Manager/400 replaced Systems Management Utilities as one of the strategic licensed programs selected to bring SystemView technologies to the AS/400 system and its users. The System Manager/400 and the OS/400 licensed programs work together through an integrated process manager to allow the user to receive notification of a problem, analyze the problem, request service (for the failing resource in the network), and provide solutions to the problem from a single point of reference.

The set of systems management programs (OS/400, System Manager/400, and Performance Tools/400) present information and provide functions in a consistent way using the Common User Access (CUA) definitions. The **CUA** interface is an SAA specification that describes the way information should be displayed on a screen and the interaction techniques between users and computers. This consistent approach and the extensive automation provided by the internal process managers enhance a user's productivity in operating a single system or a network.

For example, remote AS/400 errors cause problem log information on both the agent and managing systems to be automatically updated. An alert flows the error to the managing system, notifying the central user or operator of the condition. The central operator may then select the problem reported from the alert. This provides a view of the information about the problem and enables the central operator to remotely analyze the problem on the agent system to determine the cause of the error. Both the agent system and managing system problem logs keep up to date as to the actions taken to resolve the problem.

In this example, the internal process manager manages the functional flow by receiving the problem notification, notifying the user, providing the user with resolution alternatives, managing the system interconnection, updating the problem logs, and analyzing the problem as a single user task. The state of the problem can be found using the Work with Problems command. Additional problem information associated with the reported problem can also be viewed through this same interface on either the agent or managing system. This improves the productivity of both the agent and managing system personnel and helps to keep the user's customer satisfaction high.

This example demonstrates how many different applications and internal software components interact to provide the user with problem management control that is integrated and smooth flowing.

The AS/400 system provides functions in each of the disciplines of the SystemView technology. Examples of the components that provide functions for each discipline follow:

- Problem management
  - Problem manager
  - Alert manager
  - Message handling
  - Electronic customer support components
  - Configuration manager
  - Analyze problem (user defined and system detected) component
  - System Manager/400 problem and change manager components
- Change management
  - Distributed systems node executive (DSNX)
  - PTF manager
  - System Manager/400 PTF manager
  - Software inventory components
- Operations management
  - PC Support
  - Distributed host command facility (DHCF)
  - Display station pass-through
  - System Manager/400 remote problem analysis
- Performance management
  - Performance monitor
  - Performance Tools/400 capacity planner
  - Performance Tools/400 advisor including graphical support
- Business management
  - IBMLink (accessed through work with product information)
  - IBMLink file transfer
  - Order process component
  - Question-and-answer (Q & A) component
- Configuration management
  - APPN topology components
  - Hardware and software inventory components
  - Automatic resource configuration components

Each of these components depends upon up-to-date access to the data described in the following topic.

Data Aspects of Systems Management

AS/400 systems management data is comprised of electronic customer support contact information, routing information, alert and user notification information, configuration information about hardware and software resources, as well as performance, operations, business, change, and problem data elements.

Conceptually, the design objectives for systems management provide for automatic collection of data, allow access from many applications, cross-correlate the data where possible, and store it once. Refer to Figure 60 on page 117.

Applying this concept to the data used for systems management on the AS/400 system yields the implementation shown in Figure 61 on page 118. This data is accessible to numerous systems management functions and applications. Key data can be accessed from

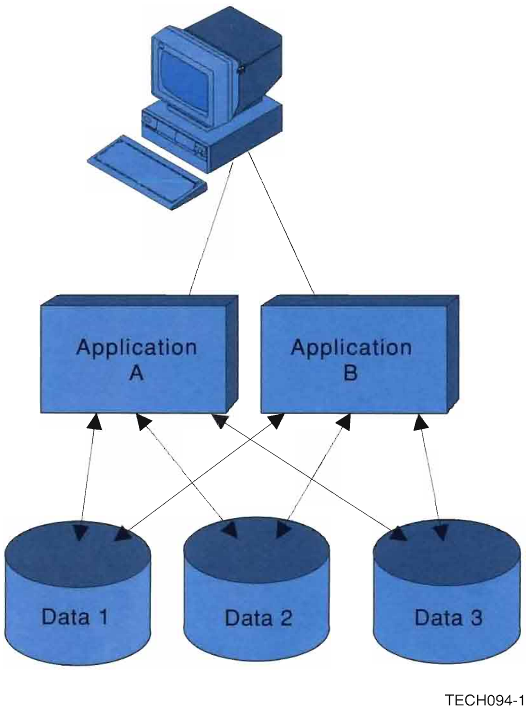


Figure 60. *Systems Management Data Concepts*

other customer-written applications as well. It is stored in a way so as to minimize replication of the data and automatically update data when a systems management task is performed. This increases user productivity through automation and reduced duplication of information. It also increases the accuracy of the electronic customer support and systems management information so that the decisions can be made from a consistent, integrated, single view of the information.

Data is automatically collected when feasible and in some instances entered or changed by the customer. Examples of key systems management data include:

- Q & A database
- PTF repositories
- Problem logs
- Contact data including service requester and provider routing information
- Alert code table (ACT) and reference code table (RCT)
- Performance data
- Hardware and software VPD and configuration data
- Message files
- Alert log and sphere of control (SOC)
- Advanced peer-to-peer networking (APPN) information

User access to systems management data extends into the **IBM support systems** (IBM service support and market support systems) through electronic customer support functions. This additional data is accessible from remote systems and provides users with information about publications, order information, and more. Examples of additional data provided by IBM support systems include:

- Product information
  - Publication order listings
  - Order Information
  - Announcement letters
  - Marketing information
  - Sales literature
  - Education information
- IBM service support systems
  - Problem management information
  - PTFs
- IBM market support systems
  - Q & A database

One of the key aspects of the data dimension of systems management is the integration and correlation of the data between data elements. The problem log, alert log, PTF repository, message files, and contact data correlate using key data to allow a total system or network view of information. Refer to Figure 61 on page 118 for some of the key data used to correlate systems management information. These data correlations along with the applications provide solutions to the user and are key to the functional flow managed by the internal process manager discussed in the topic “End-Use Aspects of Systems Management” on page 115.

Automation of data collection, multiple uses of the same data by multiple applications, correlation of data, and integrating the data across disciplines are critical factors in providing a complete systems management solution. These factors increase data accuracy, reduce redundant data management on the part of both the system and the user, and increase efficiencies and productivity of both the user and the user's customers.

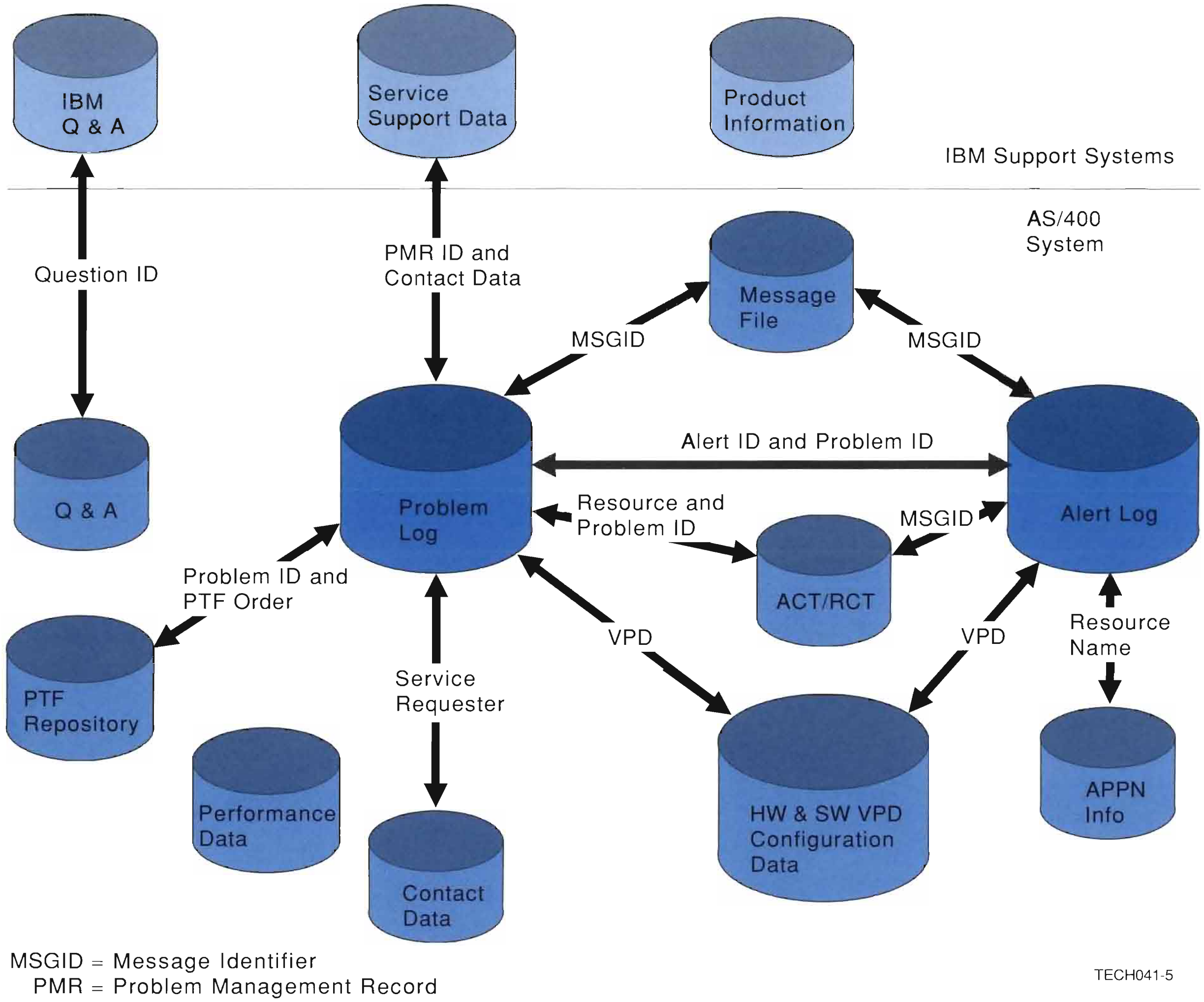


Figure 61. *Systems Management Data Correlation*

Application Aspects of Systems Management

In addition to the many changes to the data and end-use dimensions of AS/400 systems management functions, the application dimension has also been enhanced. Systems management functions are provided in the operating system.

The newest and most significant addition to the set of systems management applications is the System Manager/400. It provides extensions to the internal process manager into the centralized problem management and change distribution process.

System Manager/400, along with the operating system, no longer requires the customer to directly report problems from the system experiencing the failure to the IBM service support systems. An intermediate AS/400 system can now be configured to act as a managing system to provide service to other AS/400 systems in the enterprise network. All problems maintained in the agent system (service requester) can also be automatically tracked at the managing system (service provider) through the use of network alerts and service requests. The managing system can now create an online PTF database for its network of users as well. By creating this central repository, the service requesters can report a software problem occurring on an agent system through the functions provided by System Manager/400. The solutions (PTFs) required to solve recurring software problems are automatically sent to the reporting agent system. An alternative is to order a specified PTF from the agent system or to have the service provider on the managing system distribute a defined solution (set of PTFs) to a set of agent systems.

The systems management applications increase interoperability, ease of use, and accuracy through the use of the data and end-use dimensions previously discussed. The applications are focused on integration of the user interface, data sources, and interoperation to provide the user with new functions for resolving system and network problems and managing changes. Key functional enhancements to the applications of the AS/400 system include the following areas:

- System Manager/400
  - Ordering PTFs from a central site
  - Single action ordering of PTFs from IBM service support systems
  - Central PTF distribution to agent systems
  - Automatic PTF delivery for reported problems from an agent system
  - Problem reporting to a central site
  - Problem reporting to multiple managing systems
  - Central site service for hardware or software
  - Remote problem tracking and analysis from a central site
  - Centralized problem resolution from a central site
- Performance Tools/400
  - Performance data analysis through an advisor option
  - Communications reporting and graphical display
- OS/400 licensed program
  - PTF management and ease of use
  - Electronic customer support connectivity
  - Software resource tracking
  - Application program interfaces (APIs) to alert, resource, and problem data
  - First failure data capture
  - Operational Assistant correlation of spooled file information with problem data
  - User-defined alerts
  - Scheduled performance data collection

Each of these functional enhancements are described in the topic "Systems Management Technological Enhancements."

Systems Management Technological Enhancements

The original and current systems management products are built on key advancements, which are characteristic of the operating system. These include electronic customer support, first failure data capture, self-identification, and other items as described in the topic "Version 1 Foundation" on page 110. Since the initial release of the OS/400 product, several key advancements to the systems management functions have been made.

The initial internal process manager has been enhanced for the network environment through the System Manager/400 licensed program. The user is guided to the new functions discussed in the topic "Application Aspects of Systems Management" on page 118 from the previously existing base functions by using the problem log and logically extending this key concept into tracking not only local system problems but to tracking problems reported from the network.

PTF management has been enhanced, reducing the time required to install PTFs. It provides centralized PTF management through the System Manager/400 and simplifies the PTF order capabilities first seen in the initial release of the system to allow group ordering of PTFs from the IBM service support systems. Simplification of PTF installation through minimizing the user involvement in the IPL mode selection and notification of high-impact and defective PTFs through the electronic customer support functions also added to the repertoire of OS/400 PTF enhancements. In addition, many changes have been made to improve the fail-safe operation of working with PTFs. Examples are preventing the loading of PTFs intended for a 9406 System Unit on a 9402 System Unit, cleanup of PTF objects, changes that allow the bypass of the application of PTFs that refer to options or libraries that have been deleted, preventing test PTFs from being permanently applied, and automatically re-creating the cover letter files.

Electronic customer support connectivity with the IBM support systems has been improved through faster modems in the switched line environment and by providing a single leased line through the IBM Information Network.

OS/400 software resource tracking is enhanced to better allow user application access. For example, the Display Software Resource command and extensions to the output file support now allow additional user application access. The Display Software Resource command and the underlying software management structures have been modified to better position the AS/400 system as a key software management support system.

A key enhancement is the addition of APIs into systems management data, such as the alert log, resource data (both hardware and software), and problem log data. Additional functions also help the user to create subsets of the problem log through selection criteria, allowing the user to change the selection of problem data through the Change Problem command and through the remote problem handling provided in System Manager/400, which was discussed in the topic "Application Aspects of Systems Management" on page 118.

Time-of-failure error detection has been built into AS/400 hardware since Version 1 Release 1 (see the description of first failure data capture in the topic "Background" on page 110). In Version 2 Release 1 first-failure-data-capture (FFDC) capability is extended to AS/400 software. Using defensive programming techniques, FFDC probes are inserted into the software. The probes, using proactive error detection techniques, provide immediate notification of errors and initiate selective capture and logging of associated problem data. The data supplies information to automatically build a symptom string for the problem, create customized dumps, and build authorized program analysis reports (APARs) where necessary. The symptom string supplies the key used to do an automated search of the AS/400 PTF repository. The symptom string can also be forwarded to another AS/400 system or to the IBM service support system using electronic customer support. There, the PTF repository is automatically searched based on the symptom data, and PTFs that are found are automatically returned. This new technology reduces the need to re-create problems and shortens the problem resolution time.

The OS/400 Operational Assistant user interface provides additional first-failure-data-capture functions as well. Operational Assistant creates an entry in the problem log when the user selects the option for problem handling. Information for the current job is automatically collected and placed in spooled files. This creates a problem log entry that identifies the newly created spooled files and associates them with the problem log entry for future reference.

Alert capabilities now support user-defined alerts, allowing the user to create and send alerts defined for user-identified situations and user-provided applications. These new APIs allow user-written applications or programs to generate and send alerts to the alert focal point. There have also been enhancements in printing alert information.

A performance advisor function is now included as part of the Version 2 Release 1 Performance Tools/400 licensed program. The advisor simplifies performance analysis, directs users to specific performance problem areas, provides directed procedures for resolving identified problems, and provides users with their own knowledge-base performance expert.

The order process improvement enables users to electronically transmit their current system configurations to ensure more accurate ordering of additional hardware features and upgrades.

The OS/400 distributed systems node executive (DSNX) provides direct support. This support keeps the session with the IBM NetView Distribution Manager active until the requested operation is complete. This allows a function to be completed in a single telephone call, which has obvious benefits in a switched line environment.

These and additional new functions increase automation and provide highly layered structures, self-defining and self-diagnosing components (from hardware through applications), fail-safe exception handlers, built-in trace capabilities, programmable problem determination procedures (PDPs), and the integrated internal process managers.

Conclusion

The AS/400 system made significant advances in providing state-of-the-art solutions for managing computer systems. In Version 1 the strategic architectural foundations were built, the fundamental design concepts were established, and application solutions were provided. Architectures (such as vital product data), design concepts (such as first failure data capture and self-identifying system components), and new application functions (such as electronic customer support, automatic configuration, and integrated resource management) have established the system as an industry leader in system management as measured by the customer.

The introduction of SystemView technology placed new industry focus on system management. The SystemView focus on common user interfaces, shared use of data, and integrated and automated application program functions is consistent with the AS/400 implementation and future goals. The Rochester laboratory is committed to providing a complete set of heterogeneous management solutions on the AS/400 system. The laboratory is further committed to achieve this goal, over time, through the implementation of the SystemView structures and architecture.

Version 2 of the AS/400 system introduced significant new SystemView technology in response to growing customer needs. In the future, additional new system management solutions built on AS/400 and SystemView technology will continue to position the system as an industry leader.

Availability Enhancements

Describes software functions that improve the AS/400 system's ability to remain available to users during and after a disk-related hardware failure.

Thomas R. Crowley, Kevin A. Kelle,
Dennis R. Martin, and Michael J. McDermott

Introduction

Availability and recovery have become topics of high interest among both AS/400 customers and the IBM field force. Customer applications and systems are becoming more mission critical, and the AS/400 system is moving into markets where any system downtime is intolerable. Yet, as systems become larger, hardware failure becomes more frequent simply because there is more hardware to fail, and recovery time becomes longer. And the AS/400 implementation of single-level storage, which is a performance boon with its spreading of objects across many disk units, can sometimes become the bane of recovery.

A number of new AS/400 software functions specifically meet these rising availability requirements. Functions, such as disk mirroring, checksum protection, and disk device failure handling, improve the system's capability of surviving a hardware failure and reduce the amount of time the system is not available to process user requests. Moreover, these functions provide a variety of availability offerings to fit a wide range of customer needs and budgets.

Disk Device Failure Handling

A software program monitors the disk devices for failures and improves the availability and serviceability of the AS/400 system. The goal of this monitor is to prevent the abnormal end of system processing and the lengthy recovery associated with the abnormal end.

Failure Detection

The vertical licensed internal code controls all input/output (I/O) to the disk devices.

When a hardware error occurs on a disk device, the following actions are performed:

1. The I/O commands that are currently scheduled to be issued to the disk device are instead placed on a set-aside queue.
2. The layers of software above the machine interface (MI) continue normal paging activity to the disk unit, unaware of the hardware failure. Any I/O commands sent to the failed disk device are placed on the set-aside queue. The I/O commands to the other (nonfailing) disk units continue without any interruption. Only I/O commands to the failing unit are deferred, by placing them on the set-aside queue. Thus, those system functions that do not require information from the failing disk unit can continue to operate.
3. The licensed internal code displays a system reference code (SRC) on the AS/400 control panel to notify the system operator of the disk hardware failure. This SRC indicates which component in the disk device encountered the failure by displaying the following information:
   - Type
   - Model
   - Unit reference code
   - Location
   - Serial number
4. The licensed internal code automatically performs a series of tests on the failing disk device. The tests run periodically on the disk device without operator intervention. The system determines the operational status of the disk device based on the results of these tests. The licensed internal code that controls these tests is resident in main storage. That software is always resident to avoid a deadlock condition of trying to read that software from the disk unit that actually encountered the hardware failure.

The system remains in this condition until the recovery actions are performed.

Recovery

The service representative uses the information contained in the system reference code, which is displayed on the control panel, to repair the failing hardware component within the disk unit. The service representative performs test and verification procedures on the disk device to ensure the repair procedure is complete.

As soon as the disk unit becomes operational, the test programs being issued by the monitor program indicate that normal activity on the disk unit can resume. The licensed internal code automatically reissues the I/O commands that are held on the set-aside queue. The user tasks, which are waiting for I/O commands to complete, resume operation as the I/O operations are performed.

This availability function, performed by the licensed internal code, detects disk device failures, directs the service representative to the repair action, and automatically resumes normal operation of the system when the repair action is completed.

Checksum Protection

In the range of storage protection options, system checksum protection has the advantage of using a minimum number of units for redundant data. When a disk unit fails, the system is unavailable, but no data is lost and no data has to be restored or entered. After the service representative replaces the broken disk unit, the data is rebuilt automatically during the next initial program load (IPL). This provides a significant reduction in recovery time, compared to recovery on an unprotected system.

Checksum Theory

Checksum protection constructs sets of up to eight disk units each. An amount of storage equal to one disk unit is required for redundant data in each checksum set. This redundancy arises by the logical exclusive-OR (XOR) operation on all the disk units in a set (see Figure 62). The exclusive-OR of all the data units is called the checksum.

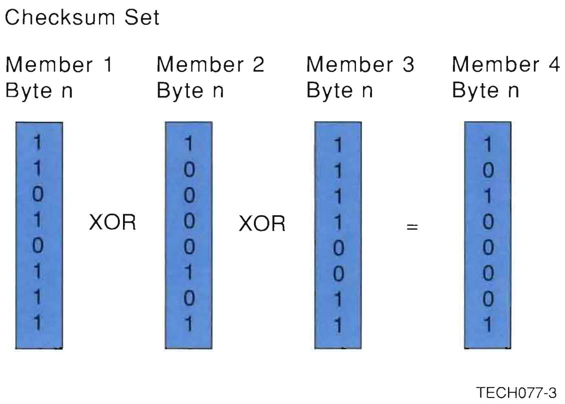


Figure 62. *A Logical View of the Checksum Concept*

If any single disk unit in a checksum set fails, the broken unit may be replaced by the service representative and the data that was on the old unit can be automatically reconstructed by the system. Because the contents of any unit of the set are actually the XOR of all the other units of the set, the data on a lost unit can be reconstructed by XORing together all of the remaining units of the set. The same principle is used to reconstruct data from an unreadable sector.

Checksum Run-Time Considerations

The update of a page of data with checksum protection in effect proceeds as follows.

1. The old data page is read.
2. The new data page in main storage is XORed with the old data page to generate a change map.
3. The page of redundant data from the unit containing the checksum is read.
4. After the old data is read, the new data can be written.
5. The change map is XORed with the old checksum page resulting in the new checksum page.
6. The new checksum page is written.

Although the read and write operations are partially overlapped, there is clearly some overhead in every checksum update (see Figure 63 on page 124).

Two mechanisms to help offset the slower performance of checksum updates are striping and the use of unprotected storage for temporary data. **Striping** is the partitioning of the disk into segments of equal size. If the checksum of a set of disk units were stored on a single unit of the set, that unit would receive more I/O requests than any other unit because the checksum has to be read and written with each update to any of the other units in the set. The distribution of the redundant data (the checksum) on different members of the

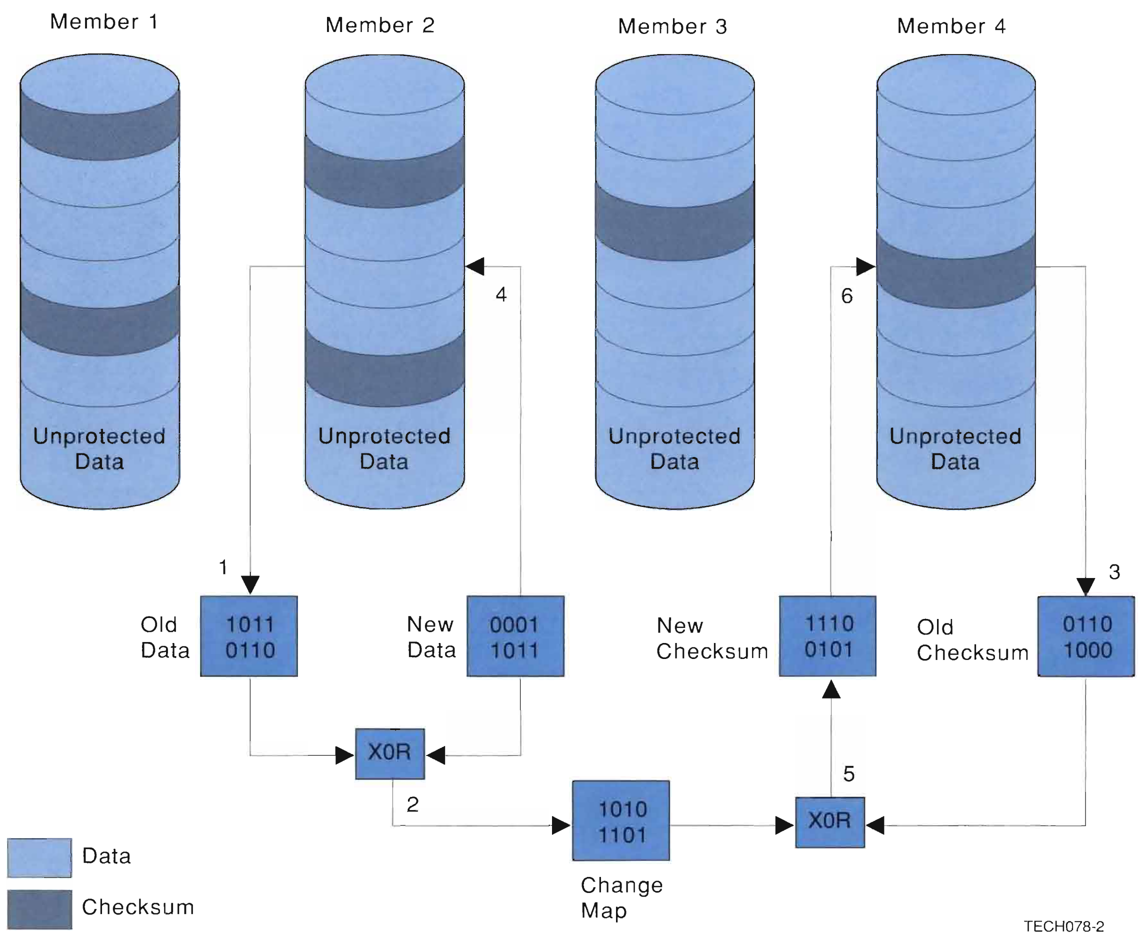


Figure 63. *Checksum Run-Time Support Showing a Checksum Write Operation*

set for different stripes eliminates this bottleneck of checksum I/O (see Figure 63).

The system requires temporary storage for use by the operating system. This storage does not have to be recovered because temporary data is re-created every time an IPL is performed. However, a significant portion of the update operations in the system are to temporary storage. The checksum update overhead is further reduced by reserving an area of temporary storage that is not checksum-protected on each unit.

Recovering from Failures

When a checksummed disk unit fails, the system ends with a system reference code, which indicates the failing unit and the type of failure. When the unit is repaired or replaced, the data is reconstructed by reading and XORing the data from all the other units in the checksum set, and writing it to the replacement unit. This checksum recovery takes place during the IPL following a disk replacement.

Single-sector read errors can be corrected dynamically without ending the system if the checksum is valid in the stripe of the disk unit with the unreadable sector.

1. The other disk units in the checksum set are read and XORed together.
2. The result is written back to the disk.
3. The result is returned to the requester of the read operation.

The sector then contains the reconstructed data, and the potentially damaged object is intact.

Mirrored Protection

Disk **mirroring** is a software function that significantly increases the availability of the AS/400 system after a failure of a disk or disk-related hardware. The mirroring support is a standard part of the Operating System/400 (OS/400) operating system, licensed program and the function can be used on any AS/400 system that contains sufficient auxiliary storage (disk capacity) and an even number of disk units of each disk type and model. To the OS/400 software, a **disk unit** is a single storage unit within a disk enclosure. Some disk types contain multiple disk units within a single enclosure.

When mirroring is activated, the system pairs disk units of the same disk type and model into **mirrored pairs**, a pair of physical disk units that together constitute one logical unit of single-level storage. From then on, the system automatically maintains the two units of each pair as exact mirror images of one another, sector by sector.

If one disk unit in a pair fails, the system can continue to run, using the copy of data on the remaining good unit of the pair. The system can survive multiple disk failures without ending, as long as at least one unit in each mirrored pair remains functional. After a broken disk unit is repaired, the system synchronizes it with the other member of the pair, which has remained active during the time of failure. The repair and synchronization can often be completed while the system remains running and available to users. During synchronization, every sector of the unit that remained active during the failure and repair is copied to the repaired unit so that all intervening data updates are captured, and both disks in the pair will be perfect mirror images of one another again. The result is an AS/400 system that can survive multiple disk failures without data loss and without a system crash.

Levels of Mirrored Protection

Level of protection is a term that describes the degree of redundancy in the hardware that exists between the data on the disk units of a mirrored pair and the processing unit. The mirrored pair is vulnerable to single points of failure in any hardware above the level of protection, and if such a failure occurs, the system becomes unusable, it must be ended abnormally, and it requires a long recovery on the next IPL. However, no data is lost.

The various levels of protection are illustrated in Figure 64 on page 126.

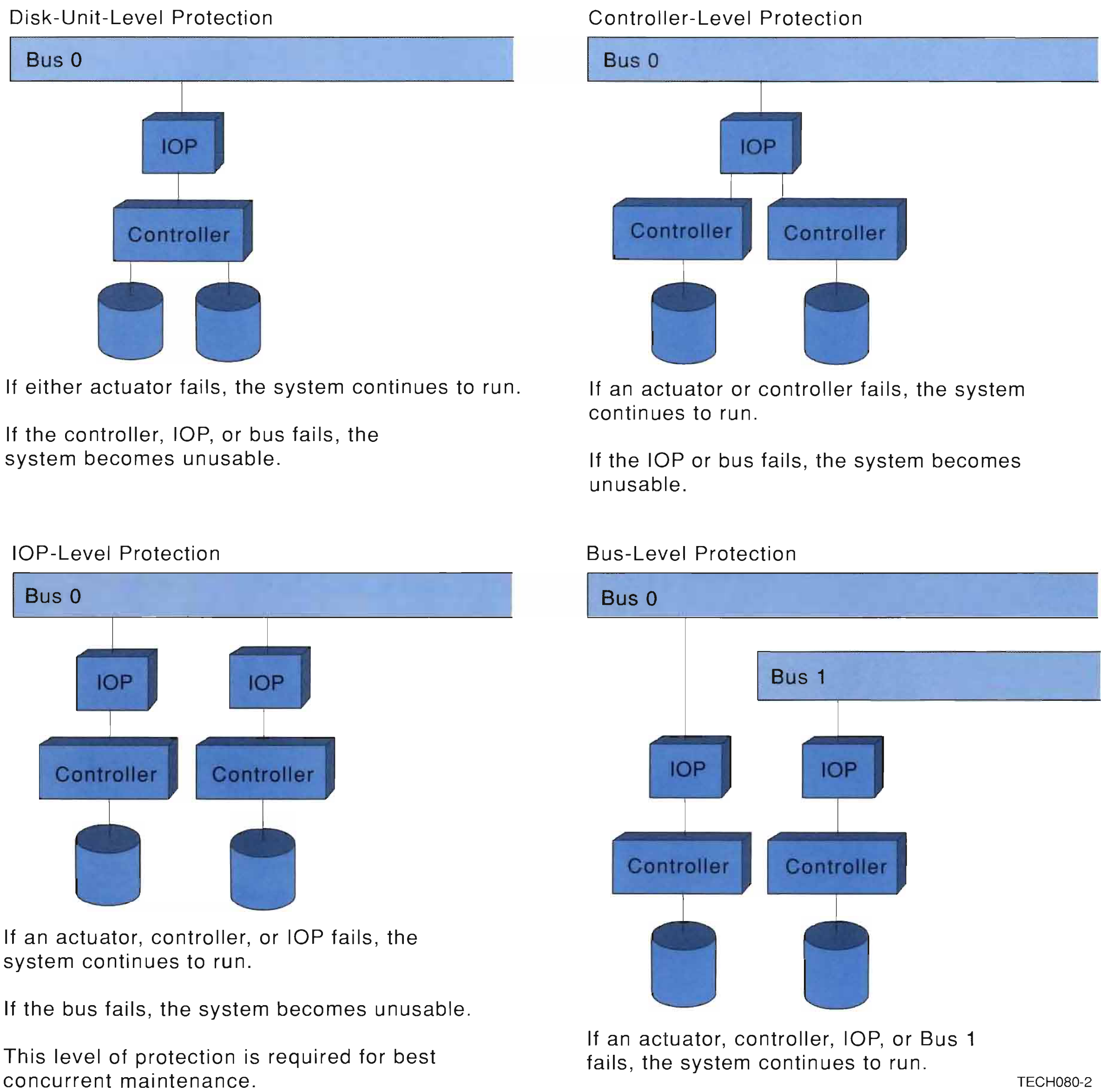
The choice of which disk units to pair with one another directly affects the level of protection of the resulting mirrored pairs, which in turn directly affects the degree of availability of a system. The AS/400 implementation of mirroring is unique in that the system automatically pairs disk units for mirroring to provide the best possible protection using the hardware available. On other systems, where a person is responsible for arranging the disk units into mirrored pairs, that person must have a detailed understanding of the physical connections and characteristics of the hardware, as well as hardware and software restrictions and interrelationships.

At best, manual pairing of disk units for mirroring is difficult, time-consuming, and error prone, and it can frequently result in incorrect configurations or configurations with less than optimal pairing and protection. In contrast, the automatic pairing on the AS/400 system provides significant availability and usability advantages.

The Pairing Algorithm

The mirror pairing algorithm is based on the concept of the value of pairing. A disk unit is compared with many potential mates and is paired with the unit that will produce the greatest possible pairing value. To determine the pairing value of a prospective mirrored pair, the logical bus addresses of the two disk units are compared. If the units are on separate buses, they have the greatest pairing value. Next greatest is separate I/O processors, followed by separate I/O controllers, and by separate disk enclosures. (In this article, a **disk enclosure** is a head-disk assembly, a self-contained, separately replaceable unit.) The two disk units of a mirrored pair are never allowed to be actuators within the same disk enclosure because the loss of one would often cause the loss of the other. The failure of both disk units of a mirrored pair produces a double failure condition, and the system cannot continue to run. The value of pairing is also improved if all of the actuators within one disk enclosure are mirrored on the actuators in another disk enclosure because the units are less vulnerable to a double failure.

The pairing and optimization avoids the brute force method of comparing all possible combinations of all disk units because the Order of



If either actuator fails, the system continues to run.

If the controller, IOP, or bus fails, the system becomes unusable.

If an actuator or controller fails, the system continues to run.

If the IOP or bus fails, the system becomes unusable.

If an actuator, controller, or IOP fails, the system continues to run.

If the bus fails, the system becomes unusable.

This level of protection is required for best concurrent maintenance.

If an actuator, controller, IOP, or Bus 1 fails, the system continues to run.

Figure 64. *Levels of Mirrored Protection*

Magnitude of such a solution is O(*n*!), where *n* is the number of disk units being paired. The current algorithms result in an Order of Magnitude of O(*n*), a significant improvement in performance, while still producing a maximum protection value.

In pairing, the system sorts the disk units by logical bus address, number of actuators per disk enclosure, and device type. Then, the system pairs devices, one device type at a time, and by actuators-per-enclosure group within a device type. A look-ahead function adjusts the starting point of pairing to optimize enclosure-to-enclosure pairing. Units that cannot be paired during the first pass become spares. If, after various manipulations of the list, the spares cannot be paired with each other, a backtracking function unpairs the existing pair with the lowest pairing value, incorporates it into the list of spares, and attempts to pair the spares again, repeating the process until all units are paired. After all units are paired, the system runs a final optimizer, which attempts to improve the pairing value of pairs by swapping their mates in various combinations. Pairing of devices is illustrated in Figure 65 on page 127 and Figure 66 on page 128.

In Figure 65 on page 127, the disk units (actuators) are represented by cylinders, and the enclosures containing the disk units are represented by boxes around the disk units. All disk units are of the same device type, all have the same number of actuators per disk enclosure, and there is an even number of disk enclosures; therefore, backtracking is not required, the look-ahead produces no changes, and further optimization after the initial pairing is unnecessary. The load source (unit 1) is omitted to simplify the example. In actual practice the disk units are sorted by device type and number of actuators per disk enclosure, as well as by logical bus address.

In Figure 65 and Figure 66 on page 128, the numbers beneath the disk units are logical I/O addresses as displayed by dedicated service tools (DST) and system service tools (SST). The labels, U3, U4, and so forth, indicate logical unit numbers and pairings.

In Figure 66 on page 128, all disk units are of the same device type, and there are an even number of actuators; however, there are an uneven number of disk enclosures in the group of devices with two actuators per enclosure. The units are paired in groups, according to the number of actuators per enclosure and beginning with the group that contains the most actuators per enclosure. In the first group, a look-ahead results in adjusting the starting point of the pairing so that devices can be paired enclosure to enclosure. Two spare units, attached to Bus 0 IOP 2, are left over when the first group of disk devices is paired, and the spares eventually result in backtracking before they are paired. When proceeding to the second group, the two single actuators of that group are paired together and then broken up to pair with the two spares during the backtracking phase. The optimization pass does not change any pairings.

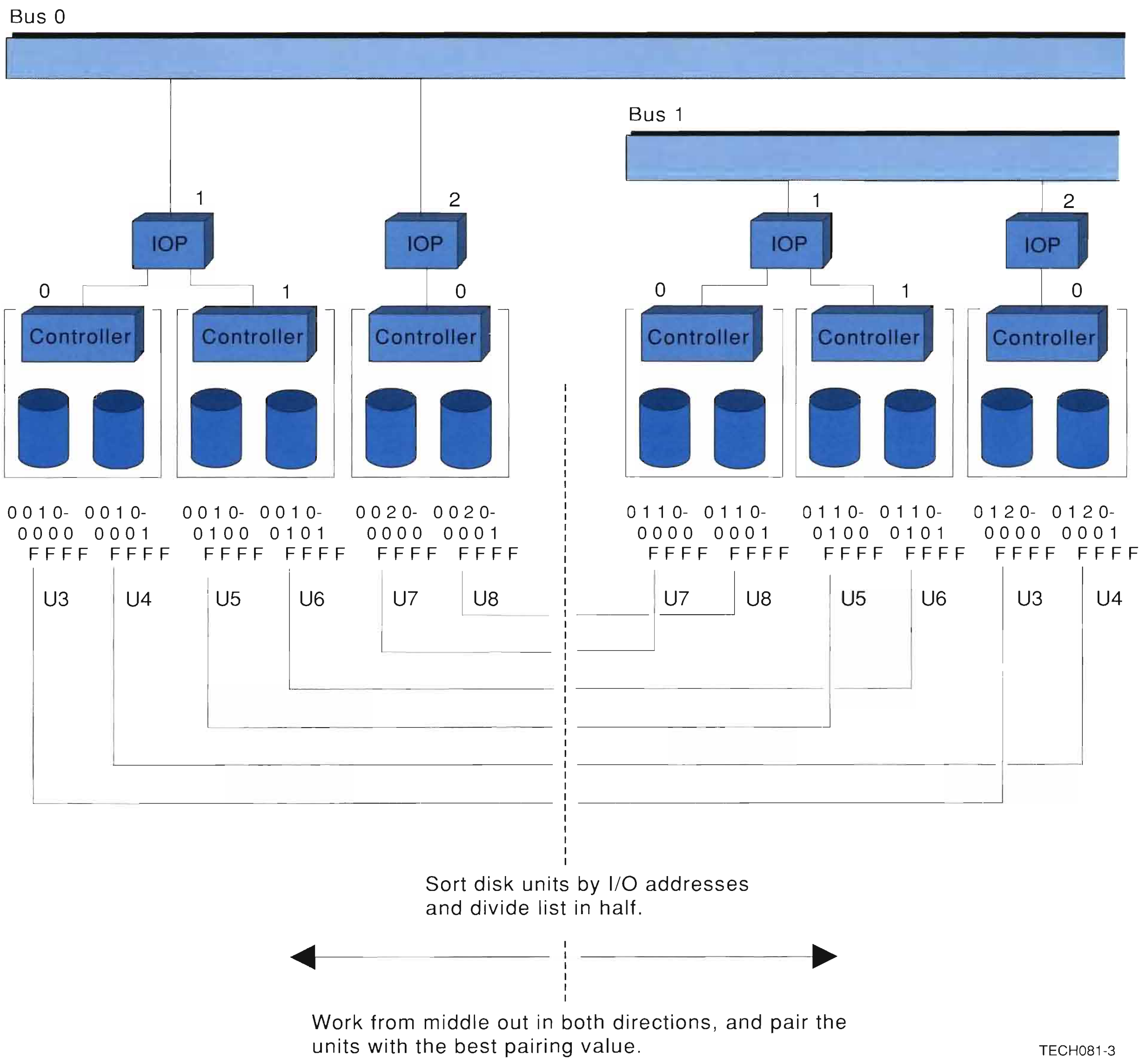


Figure 65. *Pairing Disk Devices for Mirroring—Simple Example*

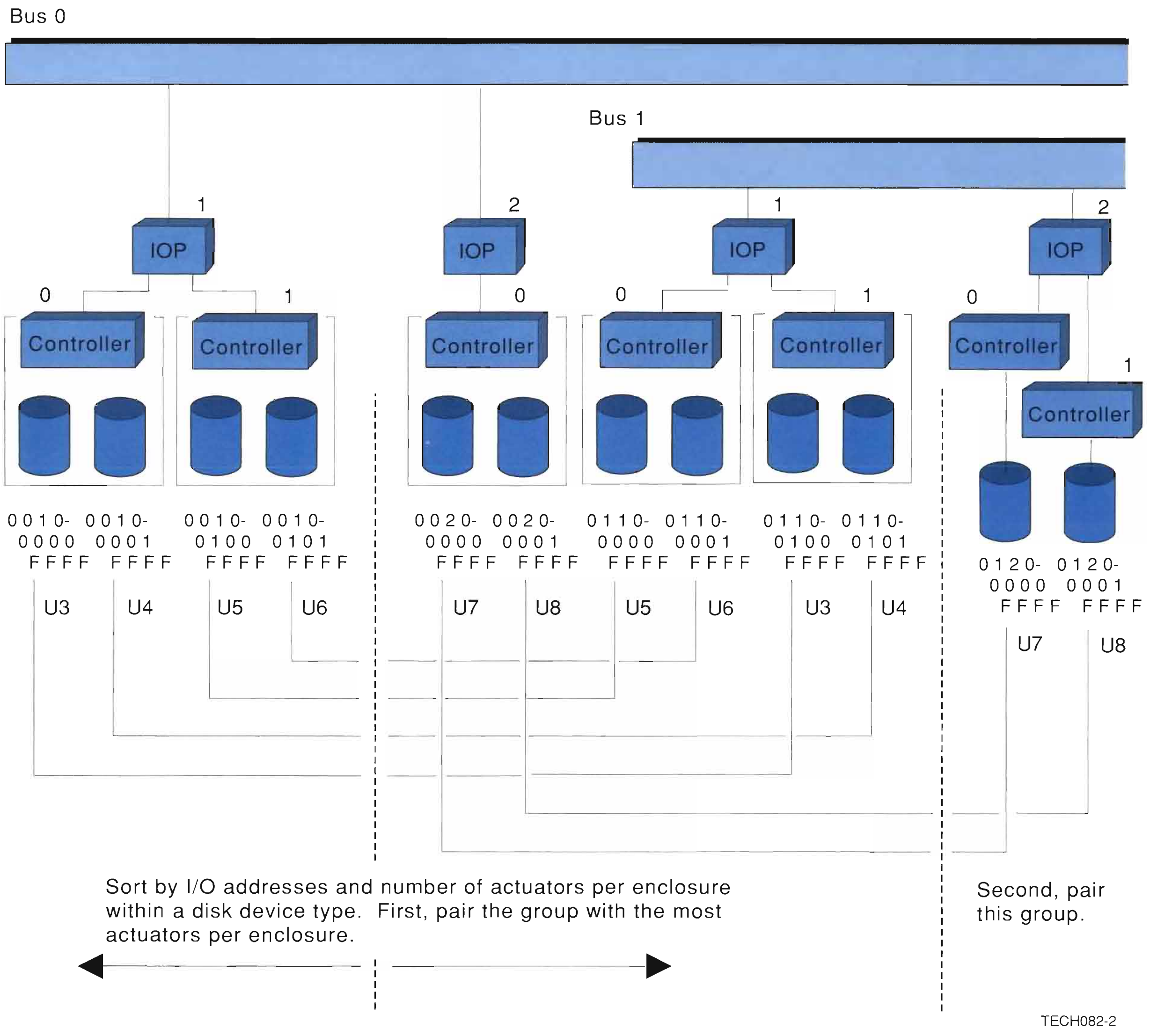


Figure 66. *Pairing Disk Devices for Mirroring—Look-Ahead and Back-tracking Required*

The algorithm is guaranteed to pair any group of devices that satisfy the minimum mirror pairing restrictions because, in a worst-case scenario, the backtracking function of the algorithm approaches the brute force method of attempting all possible pairing combinations. Manual pairing may produce a different pairing result but not one with a greater value of pairing or level of protection.

Mirror Read Optimization

The performance of read operations is optimized by selecting the unit of the mirrored pair with the faster expected response. The system performance lost by the dual write operations is offset by the improved performance of read operations. Because the ratio of read-to-write operations is typically 7:3, in some environments overall system performance is better with mirrored protection than without protection.

Selection of the disk unit with the faster expected response is complicated by the system's I/O control structure. There are several layers of software in the path from the I/O request from the operating system to the signals that drive the device. To some degree, each layer has control of the scheduling of its work. The level of control described here is somewhere in the middle, in the licensed internal code that issues the commands to the I/O subsystem. The criteria for selecting the unit of a pair from which to read does not interfere with the scheduling algorithm used by the I/O subsystem.

At the licensed internal code layer, the position of each disk actuator is maintained by saving the disk address whenever a read or write command is issued to the I/O subsystem. When a read request is received from the operating system, the read operation is issued to the unit of the mirrored pair with the fewer number of outstanding I/O commands. If both units have the same number of outstanding I/O commands, the unit whose actuator position is closer to the disk address to read is selected.

An estimated actuator position is saved on receipt of a read or write request. This is only an estimate of the true actuator position because other I/O commands may be outstanding. When an I/O command is completed, the actuator position is still unknown unless no more I/O commands are outstanding. On the completion of an I/O command with no outstanding commands, the actuator position is again saved.

Recovery from Failures

AS/400 mirroring keeps the system available when a disk-related failure occurs. Mirroring provides protection for the following failures:

- Disk device failures
- Disk controller, IOP, and bus failures if the system has the additional hardware connected such that all disk units attached to the failing hardware component have a mirrored unit attached to a different hardware component

AS/400 mirroring handles several types of disk-related failures:

- For an irrecoverable device error (for example, equipment check), the recovery processing follows:
  1. The system suspends the failing unit; mirrored protection is suspended for this mirrored pair. If the other unit of the mirrored pair is already suspended, then the system fails.
  2. The system continues operation using the other unit of the mirrored pair.
  3. The system displays a message that identifies the failing unit and informs the operator that mirrored protection is lost for this mirrored pair.
  4. The service representative repairs or replaces the failing unit using concurrent disk maintenance.
  5. The service representative resumes the operation of the repaired or replaced unit when the maintenance is complete.
  6. The system synchronizes the repaired or replaced unit with its mirrored unit.
  7. The system displays a message when synchronization is complete. Mirrored protection is active again.
- For a permanent read error (media failure), the recovery processing follows:
  1. The system reads from the corresponding sector of the other unit of the mirrored pair. If a permanent read error also occurs on the other unit of the mirrored pair, the permanent read error is not recovered; the original read request is completed with a permanent read error.
  2. If the read operation from the other unit is successful, the system writes the data back to the first unit of the mirrored pair; an alternate sector is assigned. If the write operation back to the first unit of the mirrored pair fails, the first unit now has an irrecoverable device error that is processed as previously described.
  3. If the write operation back to the first unit of the mirrored pair is successful, the system reissues the original read request to the first unit. The permanent read error is recovered and both units of the mirrored pair contain the correct data.
- For a temporary error (not operational, power failure, not ready, device time-out), the recovery processing follows:
  1. The system attempts to recover from the temporary error for a limited time (disk attention failure status or connection recovery). Any job with I/O to that unit waits during the temporary recovery attempt. If the temporary recovery is successful, normal system operation continues with mirrored protection and without suspending or synchronizing this unit.
  2. If temporary recovery is not successful within the time limit, the unit has an irrecoverable device error that is processed as previously described.
- For an IOP or bus failure, the recovery processing follows:
  1. The system determines if all disk units attached to the failing IOP or bus have

an active mirrored unit on a different IOP or bus. If not, the system fails.

2. The system suspends each disk unit attached to the failing IOP or bus. Suspending the unit is done as for an irrecoverable error.
3. The system dumps the failing IOP so the problem can be diagnosed. The system continues without the failing IOP.

Conclusion

The AS/400 system today offers a wide variety of software functions that enhance the system's availability. Some functions, like disk device failure handling, are automatic and require no explicit user action or extra hardware investment. Other functions, such as system checksum protection, require some additional hardware and protect against data loss after a disk failure, but the failure leaves the system unavailable for a time.

Mirrored protection requires twice the number of disk units as an unprotected system, plus additional IOPs and controllers, but it both protects against data loss and keeps the system running after a disk failure. With the proper hardware and configuration, broken disk units can be repaired and brought back up to date while users remain active on the system; the system can survive controller and IOP failures and continue to run.

Engineering

The AS/400 Version 2 hardware was designed using IBM's most advanced engineering processes and implemented in IBM's latest very large scale integration (VLSI), processor, main storage, disk, power, and packaging technology.

Architecture, Design, and Performance of Multiprocessors

Describes how this multitasking system, originally designed as a single processor system, was modified into a symmetric multiprocessor system.

James E. Bahr, Sheldon B. Levenstein, Lynn A. McMahon, Timothy J. Mullins, and Andrew H. Wottreng[6]

Introduction

Among the AS/400 system's unique software and hardware characteristics is the layered machine architecture [1]. The user is provided with a high-level machine interface (MI). Below this, vertical licensed internal code (VLIC) implements the operating system functions. VLIC uses internal microprogrammed interface (IMPI) instructions. Horizontal licensed internal code (HLIC) performs the operations specified by the IMPI instructions.

Many of the high-level functions performed by software in other systems are provided below the IMPI in the AS/400 system [2]. Below the IMPI, hardware and HLIC support complex functions, such as dispatching tasks to run in the system processor, queuing, and input/output (I/O) operations. These are some of the same functions that an operating system often has to significantly alter when multiprocessors are introduced in an architecture. Often the original assumptions about only one task running at a time and its altering data structures that are shared by other tasks have been violated with multiprocessors.

The unique architecture of the AS/400 system makes it an ideal candidate for multiprocessors. The operating system structure, the high-level MI, and the HLIC structure all make for a clean transition from the world of multitasking on one processor to the world of multiprocessing. The layered architecture makes it easier to introduce multiprocessors without affecting applications because multiprocessor changes can be made below the MI. In addition, the AS/400 system was already a multitasking system. Task-to-task communications and most other system structures are controlled by built-in instructions supported by HLIC. For example, the task dispatcher is written in HLIC. This type of high-level support at the IMPI level makes it possible to incorporate multiprocessor architecture with most changes at the hardware and HLIC level without significantly affecting the VLIC.

Relatively Atomic Instructions

The AS/400 multiprocessor uses relatively atomic instructions to minimize the effect of multiprocessors to the VLIC and MI. A **relatively atomic instruction** is an instruction that guarantees that while it is running, main storage locations that it uses are not changed by other processors running relatively atomic instructions from the same class. Instructions not in the same class (including instructions that are not atomic) are allowed to run simultaneously on other processors even if they access the same data. These relatively atomic instructions are divided into classes based on the type of data, called IMPI objects, that the instructions access. Instructions that are not in the same class do not access the same shared data objects, permitting instructions in each class to be atomic relative only to instructions within the same class.

In AS/400 systems, a single complex instruction can update a shared data object. For example, a single instruction can search a message queue, enqueue a new message in the appropriate position, and move a waiting task from the queue's wait list to the task dispatching queue. VLIC uses these complex

[6] This article has been submitted for publication in the *IBM Journal of Research and Development*.

instructions instead of general purpose instructions to access these shared data objects. VLIC used these complex instructions to support multitasking in AS/400 systems prior to the introduction of multiprocessors. Defining these instructions to be relatively atomic minimized the need for VLIC to add software locks.

Hardware and HLIC incorporate a lock mechanism into the relatively atomic instructions that only locks out instructions from the same class. HLIC uses a set of independent hardware locks to serialize the relatively atomic IMPI instructions and also to serialize updates to shared data areas used to communicate with HLIC on other processors. HLIC uses a different hardware lock for each class.

After one processor sets a lock, no other processor in the system can set the same lock until the first processor releases the lock. Hardware simply delays a processor's request for a lock already set by another processor until the other processor's relatively atomic instruction completes and releases the lock.

Instructions in other classes and nonatomic instructions are not affected by the lock and can operate concurrently.

The object classes follow:

- **Compare and swap** includes instructions such as COMPARE AND SWAP BYTE, COMPARE AND SWAP HALFWORD, and COMPARE AND SWAP WORD.
- **Hold record** includes instructions that provide software symbolic locks.
- **I/O** is used for I/O instructions and by the HLIC when handling I/O interrupts.
- **System timers** is used by instructions that provide time-of-day, time interval, and clock comparator support.
- **SRC** is used by instructions that provide semaphores.
- **SRQ** is used by instructions that pass messages between tasks.
- **TDQ** serializes updates to the task dispatching queue and serializes accesses by the HLIC task dispatcher.
- **Primary directory** serializes accesses to address translation tables by the translation hardware and by IMPI instructions.

Task Dispatching

The task dispatcher on the AS/400 multiprocessor provides automatic work-load balancing among processors and avoids the need for significant software changes. Similar to a single processor, all ready-to-run tasks are enqueued in priority order to a single task dispatching queue. IMPI instructions, such as those supporting semaphores and message passing that move tasks between the task dispatching queue and wait lists, still call the task dispatcher. The HLIC still performs task dispatching between IMPI instructions, but task selection is not based solely on priority. Other than initializing new task-state fields for processor eligibility, cache affinity, and the current processor, no VLIC changes are required for multiprocessor task dispatching.

The task dispatcher checks the task dispatching queue for changes since the last task dispatcher call on any processor. If the task dispatcher finds changes, it searches the queue to determine the task that should be running on each processor. If it is determined that other processors need to perform task switches, a list of pending task switches is stored in an HLIC object in main storage, and the other processors are signaled to run the task dispatcher. When the task dispatcher is called and the task dispatching queue has not changed, the HLIC performs the task switch using the information stored in the HLIC object rather than repeating the queue search. A **task switch** consists of storing the state of the current task and loading the state of the new task.

Selecting tasks is done by a combination of priority, eligibility, and cache affinity. Unless prevented by eligibility or cache affinity, the task dispatcher selects the highest-priority tasks. Eligibility can be used to restrict a task to a subset of processors and is never overridden by the task dispatcher. If all processors for which a task is eligible are assigned to higher-priority tasks, the task is not dispatched.

Cache affinity is used to dispatch tasks to the processor on which they are most likely to have residual data in cache and, therefore, the processor on which they are likely to experience the best performance.

A task is dispatched only if a processor for which it has cache affinity is available unless doing so would result in a processor remaining idle or an excessive number of higher-priority tasks being skipped. The skip threshold is specified by the VLIC. If the number of skipped tasks reaches the threshold, affinity is ignored and the task is assigned to any processor for which it is eligible. If tasks are skipped and the end of the task dispatching queue is reached before assigning a task to each processor, the skip threshold is reduced retroactively until there are either no unassigned processors or no skipped tasks.

A task initially has equal affinity for all processors. When it is initially dispatched, processor selection is based only on priority and eligibility. HLIC assigns affinity for a specific processor when the task is switched in. Certain IMPI instructions that can result in a task being removed from the task dispatching queue for a long wait can specify that the task's affinity be reset to the initial state.

Multiprocessor Hardware Support

The goals for the hardware design of the AS/400 multiprocessor system follow:

- To provide a shared main storage multiprocessor system
- To provide a bus structure to connect two processors, extendable in the future
- To incorporate a high performance protocol for main storage bus arbitration
- To provide simple serialization mechanisms to HLIC, called locks
- To handle all cache coherency problems (keeping the cache copies of a main storage location consistent)
- To provide mechanisms to handle look-aside buffer and primary directory coherency
- To have a minimal cost and design overhead in converting from one processor to multiprocessors

Two objectives identified early in the design of the AS/400 multiprocessor were:

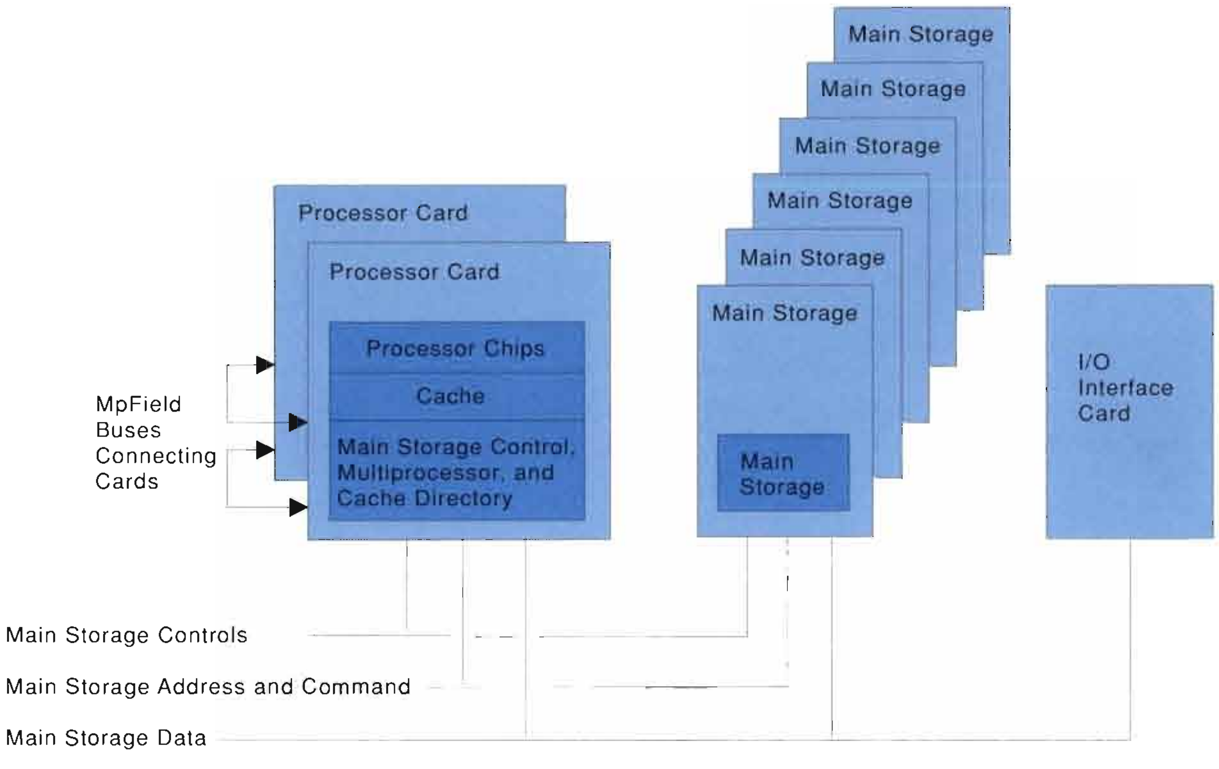
- A single design for one processor and *n*-way multiprocessors

  The goal was to create a single processor design that would require minor enhancements to produce a two-way processor. To help debug the system and further test the design, a four-way multiprocessor was the actual design point even though only a two-way processor would be the actual product. Building a four-way system in the laboratory helped expose multiprocessor bugs in the system more quickly.

- Bus snooping for cache coherency

  Because all processors are on the same system address bus and all processors do system bus arbitration in parallel, it was fairly straightforward to implement main storage bus snooping. Cache is implemented as store-through, which means that all write operations are propagated to main storage through the common main storage bus. Write operations result in other processors invalidating the cache line corresponding to the main storage location being accessed. All write operations from outside a processor are checked against a second copy of the cache directory to see if a cache line invalidation is required. Whenever an invalidation is actually required, the cache is "stolen" for one cycle to invalidate the cache directories. Invalidations are expected to have only a small effect on system performance.

In any multiprocessor, some scheme must be invented to interconnect the various processing units. The interconnection scheme affects system performance. Refer to Figure 67 on page 135 for the processor and main storage components of the AS/400 system. The AS/400 multiprocessor implementation does not have a central hub. Information that would normally be passed to a central hub is passed between processors. **Multiprocessor fields (MpFields)** are buses that are transmitted between processors. Each processor transmits on its own output bus and receives input from its peers. Logic that is based on the bus must be duplicated and kept consistent in all processors. All processors track pending write operations, fetch requests, all locks, and some other information. All arbitration boundary conditions are resolved in parallel in all of the processors.



The system planar board supports one or two processors plugged into two card slots.

TECH083-2

Figure 67. *Processor and Main Storage Components*

The AS/400 Interconnection Scheme

This interconnection scheme allows several interesting results. A single processor can run by itself or in a multiprocessor configuration. The only extra logic required is the other processors. This scheme reduces the overhead of designing a multiprocessor system. Diagnostic software uses a multiprocessor configuration register to disable processors that fail the diagnostic tests.

The bus arbitration algorithm requires several important ideas. It requires that the hub be duplicated in all processors to speed up arbitration; this includes duplication of queued write operations. It requires a special main storage card interface to allow 100% use of the main storage bus. A tiebreaker mechanism is implemented; the processor that is selected in the event of simultaneous requests changes in a round-robin fashion.

Main Storage Interface

Each processor is provided with a 128KB (KB equals 1,024 bytes) store-through cache. All processors are connected to a common main storage interface. Because the cache reduces the number of fetches on the main storage bus, most main storage bus commands are for write operations. A store-through cache puts more pressure on the main storage interface than a store-in cache, especially with many single-byte write operations.

A single bus has a limited bandwidth. Sharing a single bus results in increased latency because all processors share a common resource. Switching bus owners can be an inefficient operation. A common bus can also present cycle time problems. Ultimately, a single bus limits the number of processors that can be attached, but its simplicity is a convincing argument in its favor. Enhancements in the main storage card and processors help lessen most of the effects of a single main storage bus.

Many ideas were combined to allow one store command to be issued per cycle on the main storage bus. The arbitration protocol is efficient. There are separate address and data main storage buses. The main storage card implementation turns many write operations into read-modify-write sequences on the main storage card to handle error correction codes (ECCs). Individual bytes in main storage can be written, which allows two processors to write adjacent bytes in main storage at the same time. A main storage card can start a read-modify-write operation every five cycles; a direct 32-byte store operation requires four cycles. To lessen the effect of store operation latency, both the processors and main storage cards have a command stack. There are separate busy, card select, and data ready signals per main storage card. All processors monitor the main storage interface to allow higher utilization. In addition, if four or more cards are installed, addresses are interleaved among the first four main storage cards. (Up to six main storage cards can be installed.) Multiple main storage cards independently process different commands at the same time. As a result, contention effects on main storage latency are reduced.

Hardware Locks

A highly sophisticated locking structure is incorporated into the design. In systems with multiprocessor hardware, an extension is provided to every HLIC word. This extension provides commands used to serialize HLIC code sequences and to implement relatively atomic IMPI instructions. The hardware implements ten different locks to serialize operations on different types of system objects. High-speed arbitration allows most locks to be granted with no delay. All processors are allowed to run unless they seek to hold the same lock at the same time.

The serialization mechanism in the AS/400 multiprocessors is known as hardware locks. The hardware locks allow HLIC to serialize updates to shared main storage areas. HLIC normally uses locks to cover whole classes of capabilities like task dispatching, sending messages, compare and swap, and so forth. The hardware locks provide a way of serializing different HLIC instruction streams against one another.

Hardware locks are preferable to other serialization mechanisms because hardware locks do not involve additional main storage accesses, and hardware locks run in parallel with no cross-interference. One alternative is a test-and-set scheme, which uses main storage and operates substantially slower. A test-and-set scheme can fail, and the HLIC must handle more boundary conditions with such a scheme. Also, shared test-and-set variables increase the cache-miss ratio in many cache designs.

The ten hardware locks are independent; each can be owned by only one processor at a time. Multiple processors can own multiple locks at the same time. A given processor acquires hardware locks in a prescribed order to prevent any possibility of deadlock. The underlying hardware supports granting multiple hardware locks in a single cycle unless they conflict. Most hardware locks are granted in a single cycle. The HLIC control word following a hardware lock request can initiate a fetch request into a shared area. Releasing a hardware lock is also normally a single-cycle operation, even if there are pending write operations. All write operations that are issued while a hardware lock is owned are checked against all cache directories before another processor can obtain the same hardware lock.

The lock field is a 6-bit extension available in every HLIC control word and can be coded in parallel with any other HLIC functions.

An HLIC sequence for obtaining a lock, fetching shared data, optionally operating on the data with other HLIC words, storing data to the shared location, and releasing the lock is shown in Figure 68.

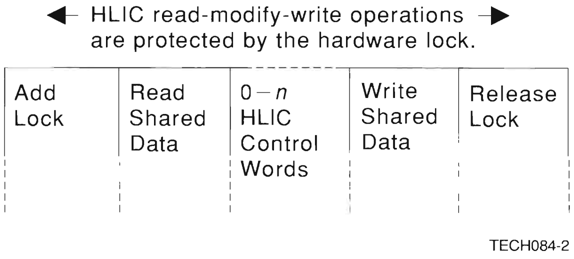


Figure 68. *HLIC Sequence for Using Hardware Locks*

Hardware locks can be added and released every cycle by all processors as long as no collisions are detected. Collisions cause the losing processor to wait until the hardware lock is released.

Simultaneous lock requests are rare but must be handled. The priority bits are kept in a pseudorandom tiebreaker register along with comparators and are used to detect and handle collisions. More often, a lock wait results when another processor is in the middle of a locked sequence.

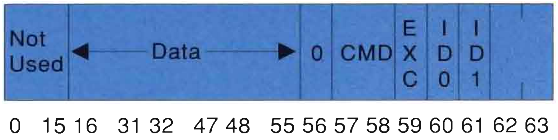
The Processor Intercommunications Register

All multiprocessor systems require a message handling scheme to send messages between processors. The AS/400 multiprocessor design uses a register called the processor intercommunications register (PIR), including an efficient set of controls to solve the problem of sending messages between processors. A message handling scheme must be capable of causing an interrupt. In the AS/400 design, that interrupt is handled by the HLIC.

The **PIR** is a hardware register that is duplicated in all processors. The PIR allows messages to be passed between processors, avoiding the cache miss that would result from passing messages in shared main storage.

Refer to Figure 69 for the PIR part of the name format. The design provides an HLIC interrupt. Some of the PIR bits are the ID (processor identification) mask; each bit corresponds to one of the possible processors. An additional bit is used to generate exceptions. When a main storage command is sent to the PIR with the exception bit on, a PIR interrupt is caused on each processor whose ID bit is set. To acknowledge a command, a processor need only reset its ID bit. Resetting an ID bit is accomplished by sending a main storage command to the PIR with the exception bit off.

The PIR acts like a main storage location and can be protected with hardware locks. Data written to the PIR of one processor is also written to the PIR of each of the other processors. No lock is required to reset an ID bit because the hardware can reset the bit atomically. The PIR ID bits also serve as busy indicators. Busy indicators and hardware handling of them allow a processor to process other IMPI instructions after sending a message.



Data: 40 bits, enough for a virtual address segment identifier (SID) and page identifier (PID)

CMD: 2-bit command

EXC: 0 = Reset acknowledgment bits only
1 = Write command, data, and ID bits to all processors' PIR and raise multiprocessor exception on processors with ID bit = 1

ID0-ID1 = ID bits for processors 0-1, respectively

TECH085-3

Figure 69. *PIR Format*

The data field in the PIR is large enough to send a virtual address segment identifier (39 bits) in bits 16 through 55. This field width allows the PIR to send the virtual address of a page to be purged from the look-aside buffers of all processors. The data field is also used as a command extender; the CMD field is only 2 bits.

Hardware Primary Directory Lock

The AS/400 system uses virtual addresses. Such addresses must be translated to real addresses before accessing main storage. High-speed look-aside buffers contain the most recently translated addresses in the processor itself. Special hardware allows maintenance of the primary directory and look-aside buffers.

The primary directory is the main table used in translating virtual addresses. The primary directory resides in main storage and is shared by all processors and all tasks. Look-aside buffer misses result in a search of the primary directory by the hardware. The primary directory lock, used by the HLIC to serialize updates to the primary directory, also waits for the completion of any hardware primary directory searches and blocks the start of new searches, making the hardware searches atomic relative to HLIC updates.

HLIC follows special rules in turning off the valid bit associated with a virtual address in the primary directory. If the valid bit is reset, mapping of the virtual address to the main storage page is ended, and any subsequent translation of the address causes a page fault. The reference bit must be off to safely turn off the valid bit. If the primary directory lock is held and the reference bit is off, the valid bit may be turned off. If the reference bit is on, a PIR message is required to purge the page from all of the look-aside buffers before the valid bit is reset.

Set Bit Function

The main storage card provides a set bit function. This function is a special command that turns into a read-modify-write operation at the main storage card. Therefore, the hardware in the processor does not have to do a read-modify-write to set the reference and change bits during a virtual address translation using the primary directory. This set bit function allows multiple concurrent translations.

Performance

The performance objective for the two-way multiprocessor (Model D80) is 1.7 times the Model D70 single processor. This performance objective represents a 15% degradation from the ideal performance factor of two times a single processor.

An interactive work load called RAMP-C (Repeatable and Measurable Performance—COBOL) provides an environment to model and measure the performance of the AS/400 multiprocessor system. RAMP-C is not representative of a specific customer's environment. Therefore, other environments may cause a significant departure from the performance values given.

Two phases make up the performance analysis effort for this design:

- Modeling the hardware and software design prior to completion of hardware implementation
- System performance measurement

Hardware and Software Design Modeling

Various analysis techniques serve to assess aspects of AS/400 multiprocessor performance. Two classes of performance factors require evaluation: 1) hardware and HLIC effects and 2) software implications. The hardware and HLIC performance evaluations arise from concepts like:

- Cache coherency
- Cache-miss rate, both aggregate and dynamic
- Main storage contention
- Synchronization of instruction class locks
- Look-aside buffer synchronization
- Queue structure manipulation

Several models provide for assessment of the performance questions at different levels of complexity. These are:

- A simulation model, which allows for an understanding of main storage contention and the effect of cache coherency maintenance. This model is a low-level abstraction of the processor design. It encompasses all queuing points in the hardware and is driven by sequences of IMPI instructions. Pseudo-HLIC strings emulate the instructions to provide a detailed interaction of machine units just as in the actual design.
- Analytic approaches, which provide results for the hardware synchronization mechanisms and provide estimated effects of longer waiting lists for queues. Spreadsheet models are common for these efforts.
- Address trace analysis, which is important to help understand cache performance questions. Many real address main storage traces taken from AS/400 Model B70 machines are evaluated against models of a cache directory. Reference patterns and overall behavior in terms of the cache-miss rate are outputs of these models.

One of the major design bottlenecks to multiprocessor throughput is the main storage card utilization. Study of the model results show a significant sensitivity to the number of main storage cards installed. For card utilization above 35%, aggregate throughput of the multiprocessor drops off rapidly as shown in Figure 70 on page 139. The two-way multiprocessor card utilization averages 22% at peak processor throughput.

The combined effect of the hardware and HLIC evaluation realizes a 7.5% degradation from an ideal performance factor of two times a single processor. This evaluation forms the basis for additional study of performance effects due to software implementations.

A software simulation model allows for an understanding of multiprocessor operating system effects. The performance model simulates the flow of tasks due to the operating system and RAMP-C users. Cache affinity (or the ability to maintain tasks on the same processor to maximize cache-hit rate) can be observed by the model. The effects of different algorithms for allocating tasks to processors is a key element of this area of the analysis. In particular, the skip threshold pro-

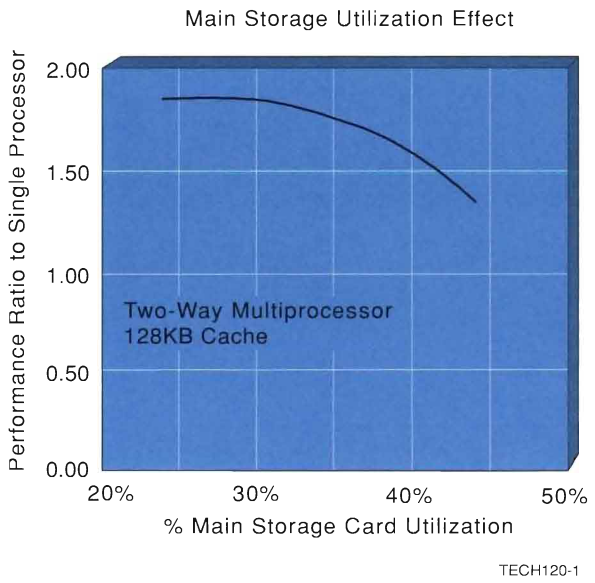


Figure 70. *Processor Sensitivity to Main Storage Card Utilization*

vides a means of limiting the search of ready-to-run tasks. Preference is given to tasks with affinity for an available processor. However, no more than the skip threshold number of tasks can be bypassed when evaluating the ready list.

Modeling portrays the sensitivity of the cache affinity of tasks to different skip threshold values as shown in Figure 71. Increasing the skip threshold provides a greater probability of finding a task with affinity for an available processor. As can be seen in the figure, a skip value of 1 provides the majority of benefit. Little additional probability of finding a task with affinity is gained for increased searches. The performance price paid for a larger skip threshold also becomes a poor trade-off.

Modeling results show a multiprocessor degradation of 2.2% due to all software effects. The total hardware and software degradation is 9.7%. Starting with an ideal multiprocessor ratio of two times a single processor, the modeled degradation yields a multiprocessor performance factor of 1.81 for the two-way multiprocessor machine.

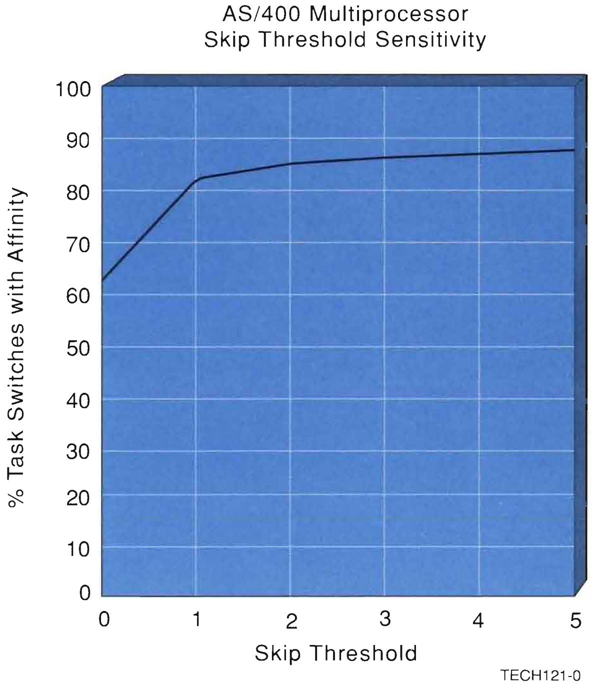


Figure 71. *Skip Threshold Summary*

Performance Measurement

To measure multiprocessor performance, two sets of internal performance counters are employed in the hardware. The first set of counters accumulates instructions, cycles, and frequencies for 21 unique states of the multiprocessor. These states include run-time instructions, wait states, exception states, and multiprocessor hold-off states. The counters are accumulated in a reserved section of main storage. A second set of internal counters monitors cache hits and misses, main storage bus use, and multiprocessor lock conflicts. Measurements undertaken on systems running the RAMP-C benchmark show a combined degradation of 10.7% from the ideal multiprocessor. This yields a degradation in the multiprocessor performance factor from an ideal value of 2, down to 1.79 times a single processor. Table 2 summarizes the performance model results and system measurements for the AS/400 multiprocessor.

Results	Realized Performance	Ideal Performance
Model	1.81	2
Measurements	1.79	2

Table 2. AS/400 Multiprocessor Performance Summary

Conclusion

The AS/400 multiprocessor effort represents a significant step forward for the AS/400 system. The goals of the multiprocessor architecture were to provide a single design point that supports a single processor and a two-way multiprocessor, and to minimize the software changes required. The operating system was not significantly changed to handle shared data objects in a multiprocessor environment because the code already used either the relatively atomic instructions or other serializing mechanisms based on the relatively atomic instructions to handle shared data in a multitasking environment. Providing high-level relatively atomic instructions had much less effect on the software than if the conventional approach to interlocking instructions were used.

In addition, the design supports a four-way processor used in the development laboratory so that multiprocessor problems surfaced more quickly during development. Another design goal was to provide HLIC with the necessary mechanisms to efficiently implement AS/400 multiprocessors. The design maintains both simplicity and performance with minimal overhead.

Performance modeling of the AS/400 multiprocessor yields a better understanding of the task dispatching characteristics of the system and hardware effects, enabling HLIC to modify the task dispatching algorithm and to ensure that the multiprocessor met its performance objectives. The performance models produce a performance factor of 1.81 times a single processor. System performance measurements yield a factor of 1.79. Thus, there is only a 2% difference between model results and measurements.

References

1. Funk, M.R., Schmierer, Q.G., and Thomforde, D.J., "System Processor Architecture," *IBM Application System/400 Technology*, SA21-9540, 100-103. June, 1988.

2. Clark, B.E. and Corrigan, M.J., "Application System/400 Performance Characteristics," *IBM Systems Journal*, Volume 28, Number 3, 407–423. 1989.

System Main Storage

Describes the function and implementation of the AS/400 intelligent main storage cards.

Richard G. Eikill and Randall S. Jensen

Introduction

The AS/400 intelligent main storage is a critical component of the *n*-way multiprocessing architecture (for a complete description of the *n*-way architecture see the article "Architecture, Design, and Performance of Multiprocessors" on page 132). To optimize system efficiency, this new main storage architecture performs many tasks that were previously done by the system processor. Moving these tasks to the main storage cards allows the system processor more time to spend processing data and eliminates a system bottleneck when multiple processors use common system main storage.

The main storage cards communicate with the system processor over a synchronous high-level command-driven interface. This command-driven interface allows the system processor to quickly notify the main storage that a task must be performed and to pass any required data. This type of interface increases the usable bandwidth of the main storage bus and allows for multiprocessor growth.

The term **intelligent main storage** refers to the processing power that exists on the main storage cards. This processing power includes system processor communications, data manipulation, data buffering for multiple commands, dynamic random-access-memory (DRAM) control and maintenance, and main storage diagnostic tasks. The advantage of intelligent main storage cards processing data is magnified by the fact that all main storage cards installed in a system can actively process data in parallel. This is a significant performance advantage.

The processing capability of the Version 2 main storage is the key distinguishing factor between this and previously released main storage architectures.

Architecture

The architecture is best described by looking at the main storage interface first, followed by an overview of the processing capability of the cards. A more detailed description of the actual hardware implementation is also included after the interface and processing capability are described.

Main Storage Interface

The main storage interface is described briefly in the topic "Main Storage Interface" on page 135 in the article "Architecture, Design, and Performance of Multiprocessors" on page 132. Some additional information is provided here.

The interface from the system processor to the main storage cards consists of a 37-bit command bus, an 8-byte data bus, and some miscellaneous control lines. This is a synchronous interface, which is designed as part of the system processor. The control of this interface is distributed among all system processors and main storage cards present on the main storage bus. The multiprocessor communications scheme used by the system processors allows the same decisions related to main storage access to be made by all system processors every cycle. This parallel processing of access requests is an efficient means of controlling the main storage interface.

See the topic "The AS/400 Interconnection Scheme" on page 135 in the article "Architecture, Design, and Performance of Multiprocessors" on page 132 for more information about this multiprocessor communications scheme.

While the system processor controls this interface under most circumstances, the main storage cards control it after they have been selected by the system processor to return data. When the data transfer is complete, control returns to the system processor. This interface is designed to optimize the data transfer rate to all system processors and to allow for 100% use of the main storage bus.

The internal input/output (I/O) bus is also attached to this interface. The system processor controls the interface during transfers between the main storage cards and the internal I/O bus. Data is passed directly between the two devices during these transfers, allowing for improved performance as pages of data are moved between auxiliary storage and main storage.

This interface is shown in Figure 72.

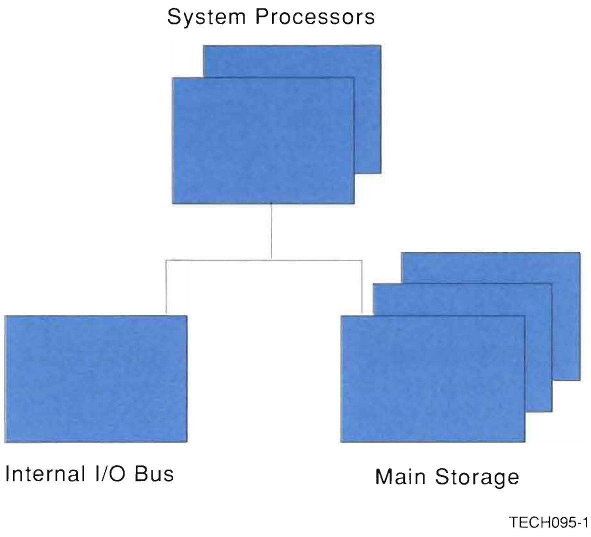


Figure 72. *Shared Main Storage Interface*

Processing Power of Main Storage Cards

The system processors take advantage of all main storage cards installed in the system by keeping multiple cards active in parallel. Each main storage card acts as a separate processing unit. The cards respond to commands from the system processors and release control of the interface while processing these commands. This allows any one of the system processors to issue another command across the interface or receive previously requested data from any one of the main storage cards. The main storage cards can also accept multiple commands and overlap processing of these commands to take full advantage of the performance functions of the DRAM devices.

The system processor implements an addressing scheme that allocates address space across four main storage cards of the same size installed in the proper physical locations. The address space on these four cards increments on a 512-byte page basis from one card to the next, rather than from the beginning of a card to the end of it. This card interleaving is intended for a multiprocessor environment to reduce the contention that results from multiple accesses to the operating system that is resident in the lower address range of main storage.

Hardware Implementation

The main storage cards handle system processor communications by responding to commands from the system processor. These commands include requests to return main storage data to the system processor, store new data in main storage, change existing main storage data, and return internal register information. The main storage cards also have the capability of informing the processor of various error conditions that may have occurred either within the hardware on the card or with the data stored on the card.

These functions are implemented in hardware by using two main storage processor chips. Each chip has a 4-byte data path to the processor and an 8-byte data path to the DRAM devices. These two chips are actually the same design while one acts as a primary and the other as a secondary processor. All functions are performed in parallel on these two chips although only the primary chip reports error conditions back to the system processor.

Each main storage card is capable of accepting up to two commands from the system processor and overlaps processing of these commands. This overlap is designed to take advantage of some of the fast access functions of the DRAM devices.

The main storage architecture is optimized for a multiprocessor system with cache support, which implies that most data transfers from main storage to the processor are of a length equal to the cache line size. This main storage card does, however, support a system with no cache. The system processor can request data in 8-byte multiples up to a total transfer length of 64 bytes. When data is moved from main storage to the processor, main storage accesses 16 bytes of data at a minimum. The DRAMs are architected in a four-bank configuration (a **bank** is a partition of main storage) that allows bank interleaving for maximum performance (see Figure 73 on page 144). Each main storage processor chip maintains the control lines of two banks and accesses the data in all four banks.

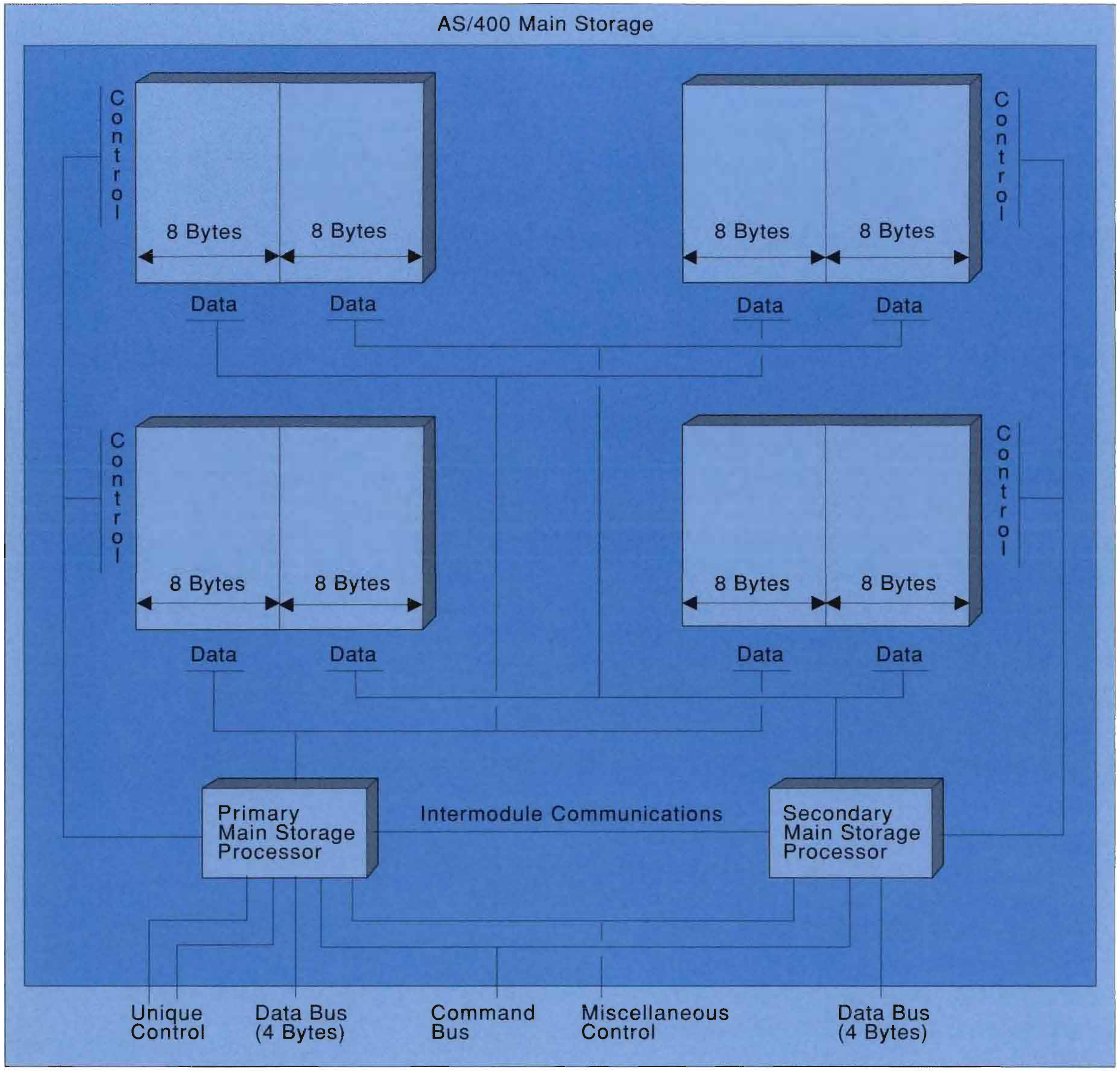


Figure 73. *Main Storage DRAM Bank Structure*

Main storage can be written in bytes to allow two processors to write adjacent bytes in main storage at the same time. The main storage implementation turns these write operations into read-modify-write sequences on the main storage card so that new error correction code (ECC) data can be generated. When data is stored in multiples of 4 bytes, a new ECC pattern is generated, and a direct store sequence is performed, which is a faster operation than the read-modify-write.

The main storage cards can change main storage data, eliminating costly data transfer operations between the system processors and the main storage cards. The system processor can set or reset specific bits in a data field in a single read-modify-write sequence on the main storage cards by issuing a single command. In addition to this, the AS/400 architected tag bits for any page of data can be inserted or extracted by the main storage cards. This read-modify-write operation is used by storage management to handle the AS/400 architected tag bits associated with data as it is moved between auxiliary storage and main storage.

The internal register information that may be passed from the main storage processor chips to the processor consists of information related to any data errors that may have been encountered. The main storage processor chips maintain all ECC data and correct any user data errors found when an access is performed. The main storage processor chips then save information related to which bits were corrected at which main storage location and notify the processor that an error has been corrected. The system processor may then request this data at a later time so that it can take appropriate storage management actions.

The main storage processor chips also save information relative to the state of internal hardware sequences and pass this information back to the system processor when requested.

In addition to the above tasks, the main storage processor chips generate all DRAM address and control signals. The main storage processor chips interpret the location of the data sent from the processor and generate the proper control signals to access this data. Also, the main storage processor chips handle the DRAM refresh operations with no interaction from the system processors.

One other function of the main storage processor chips that is not specifically related to improving system performance, is built-in diagnostic routines that verify the integrity of all DRAM devices present on the card. This function is used to help reduce initial program load (IPL) time.

Technology

Up to six main storage cards can be attached to the main storage bus, providing up to 384MB (MB equals 1,048,576 bytes) of storage in the Model D80 system (see Figure 74). The cards are enclosed in a book package, which is latched into a card enclosure. The cards are 229 by 279 mm and contain eight layers for power distribution and signal wires. Two connectors provide a 246-pin interface to the main storage bus for power and signal pins.

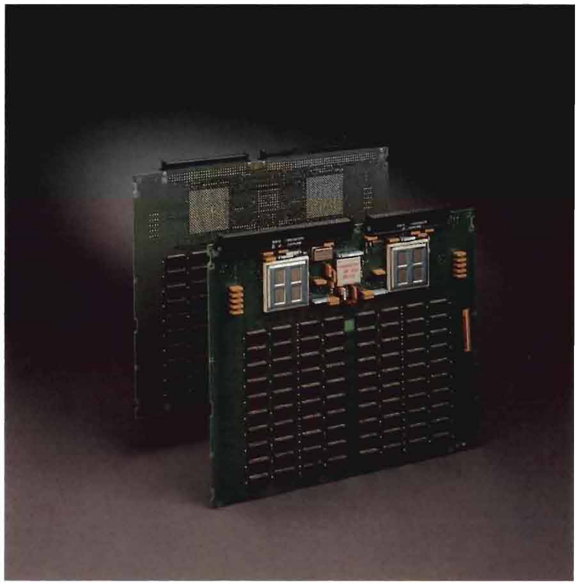


Figure 74. *64MB Main Storage Card*

High density packaging is achieved by using 4Mb (Mb equals 1,048,576 bits) memory chips (see Figure 75) in a surface-mount module. The memory modules are 17.8-by-8.9 mm with J-shaped leads. These modules are placed on both sides of the 64MB card, 80 on each side. The 4Mb memory chips are organized as 1Mb deep by 4 bits wide. Data access time of the memory chips is 80 ns.

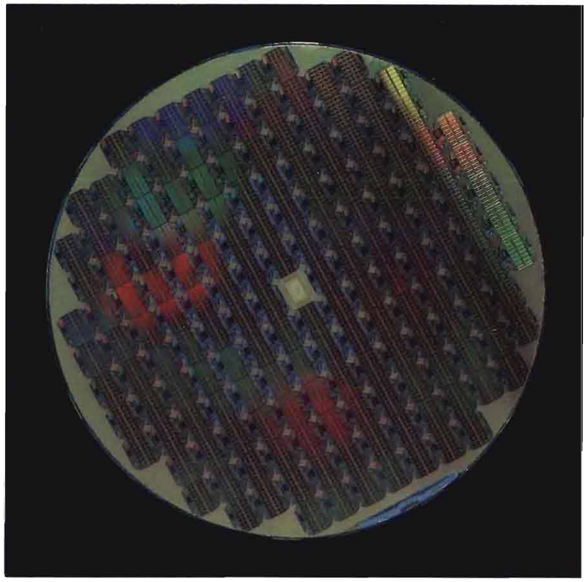


Figure 75. *4Mb Memory Chip*

Advanced complementary metal-oxide semiconductor (CMOS) technology is used for the two main storage processor modules. State-of-the-art 0.5-μ, triple-level metal, 12.7-by-12.7-mm chips provide up to 75,000 equivalent two-way, logical AND with inverted output (NAND) circuits. The nominal circuit delay of a two-input NAND driving two loads is 0.4 ns. A self-test function uses random patterns to test the chips each time the system is started. Delay compensation circuitry controls the transition time of the signals from the modules to minimize delay variations caused by variations in temperature, voltage, and chip process parameters. The chip is placed on a multiple-layer, ceramic substrate using a solder ball attachment process. Decoupling capacitors on the substrate provide more efficient noise protection by being closer to the chip. The chip-to-module pin wires are customized using eight layers within the substrate. This 44-mm module has 545 pins on a 2.54-mm grid with an interstitial grid pattern.

The main storage bus and the main storage cards operate at a 45-ns clock cycle. A primary clock is sent to a clock generation module on each card, which then generates the clock signals used by the control logic. This clock generation module uses a digital tuning circuit for precise control of the clock signals.

Future enhancements to this main storage card include the use of 16Mb memory chips in a surface-mount module. This expands the storage capacity to 128MB using 80 of these modules placed on one side of the card (see Figure 76).

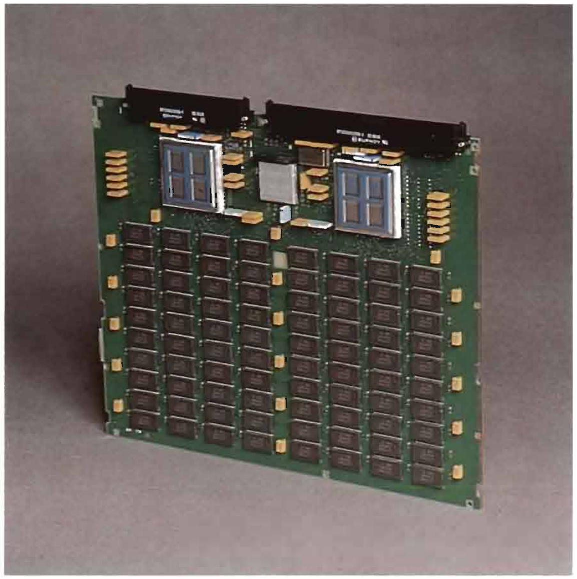


Figure 76. *128MB Main Storage Card*

Conclusion

The AS/400 main storage has advanced the state of the art through architecture and technology improvements. This is the most technically advanced main storage card ever designed for the AS/400 system. It now plays a key role in system processing.

System Processor Technology

Describes the chip technology, the design of multichip interface drivers, and the clock generation and distribution of the Version 2 processors.

Dennis T. Cox, Charles L. Johnson,
Bruce G. Rudolph, and Robert R. Williams[7]

Introduction

One of the requirements for a new processor design is to increase performance over predecessor designs. The increased performance is achieved by using faster logic circuits, wider data paths, larger arrays, and by using multiple processors. The wider data paths require that more signals be switched simultaneously. The variations in the manufacturing process, voltage applied to the circuits, and operating temperature greatly affect the switching time of the off-chip drivers. It is the switching time and the number of signals switching simultaneously that can cause noise to be generated. To limit the noise, circuits control the slope of the off-chip drivers under all of the variations that are expected to be encountered.

To get predictable results with the faster logic circuits, it is necessary to keep all of the clocks of all the chips synchronized. The system clock skews are minimized through a distributed phase-locked loop and an automated procedure to balance clocks on the logic chips.

This article describes the semiconductor and packaging technology and the design of the off-chip drivers, clock generation, and distribution used in the Version 2 processor and main storage cards.

Semiconductor Technology

The logic technology used for the AS/400 processor design was an 0.8-µm complementary metal-oxide semiconductor (CMOS) standard cell. It best fit the requirements of performance, density, power, and design closure. To keep the design cycle as short as possible, the chip designs had to be predictable from the high-level language down to the chip's physical design. This predictability allowed the overlapping of many of the design steps since each step was not strongly dependent on the preceding one.

The 12.7-mm chips are configured as a channeled array of 94 rows by 608 columns with 35 wiring tracks between each back-to-back row pair. The horizontal metal wiring is on a 3.2-µm pitch while the vertical has a 4.4-µm pitch. The third level metal is used exclusively for power busing and pad transfer for the perimeter input/output (I/O) circuits. Typically, 51K (K equals 1,000) of the 57K cells can be wired producing a density of 75K equivalent two-way AIs. The number of transistors per chip ranges from 200K through 440K, depending on the amount of array macros.

The nominal performance of 0.4 ns for a two-way AI driving a fanout of two and 0.2-pf wire is achieved with a 0.5-µm channel length device. The 3.6-V power supply coupled with 15 to 25% switch factors produces power dissipations of 1 to 2 watts. The power dissipation also varies depending on how many of the 328 I/O are used as outputs and the type and loading of those outputs.

There was an extensive list of logic library elements from which to choose. The base AI/OI/A/O logic books were available with up to nine inputs. A variety of AO/OA/AOI/OAIs with up to eight sets of two inputs could also be

[7] © 1991 IEEE. Reprinted, with permission, from *Proceedings of IEEE International Conference on Computer Design*; "IBM AS/400 Processor Technology"; Boston, MA, October 14 through 16, 1991; 448–452.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the IEEE copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Institute of Electrical and Electronics Engineers. To copy otherwise, or to republish, requires a fee and specific permission.

selected. There was an assortment of complex books, such as latches, registers, compares, decodes, parities, and multiplexors. Single-port arrays could be built in various configurations up to 18Kb (Kb equals 1,024 bits), with access times from 6 to 9 ns being worst case. Two-port arrays, up to 9Kb, could also be built with the same access times. Two 4-port general purpose registers (GPRs) were made available in 8-by-18 or 16-by-18 formats with performance in the range of 5 to 7 ns being worst case.

Design closure is the ability to reasonably predict the results in each step of the design process to facilitate overlap and result in shorter overall design cycles. This was designed into the technology in several ways. Each book had a number of output drive levels that could be automatically selected with output load-driven crossover points. This allowed the delay to be held fairly constant across a range of capacitance values and helped compensate for variations in the capacitive loading estimates due to placement and wiring of the chip (see Figure 77). Each drive level of the book fit in the same amount of cell area, which meant that powering up books could be done without disturbing the wiring of the books on the chip. The fixed cell area kept the powering up from being an iterative step.

The macros on the chip were also designed to facilitate design closure. They were designed to be porous to the wiring planes instead of totally blocked. Thus, their placement has little effect on the overall capability of chip wiring and can be done automatically along with all the other books on the chip. Another design closure aspect was the gate-array backfill in whatever spaces were not used by the standard cell books. Because of the wire density limitations on the chip, there were quite a few of these spaces. These gate-array backfill books allowed many logic updates at the metal level, thus prevented having to reprocess the whole chip from the beginning.

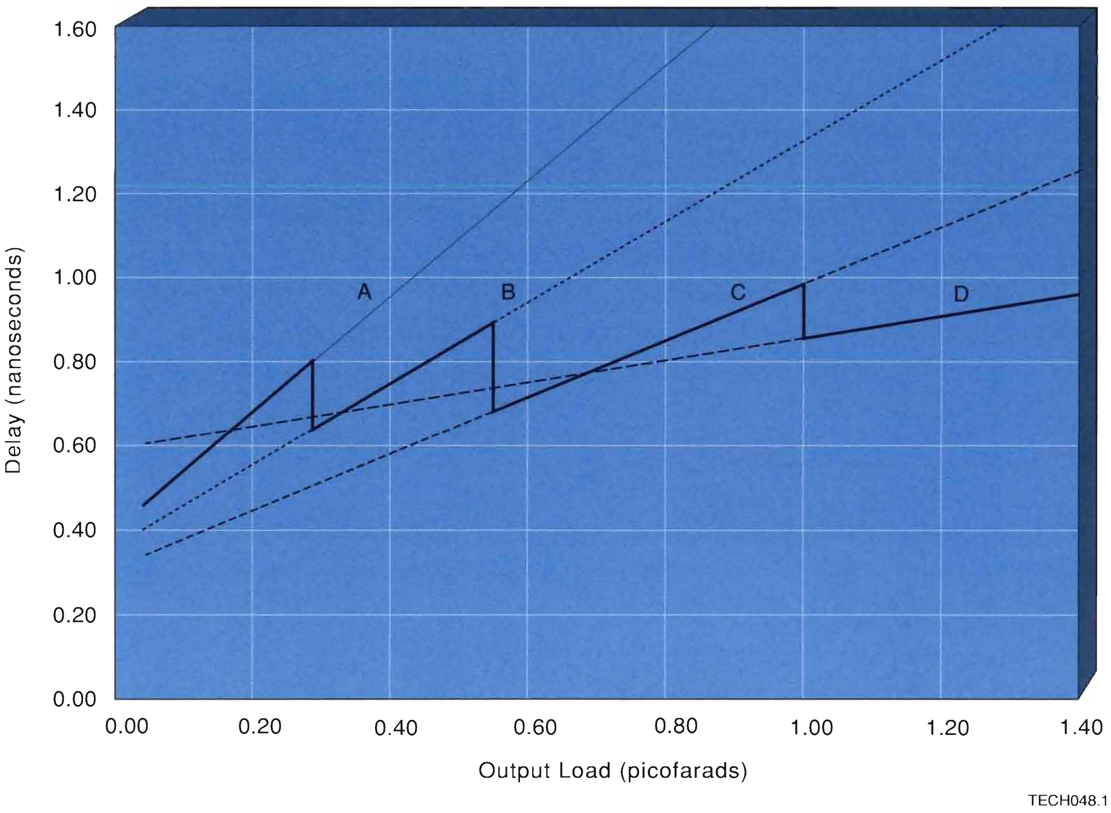


Figure 77. *Load-Driven Crossover Point*

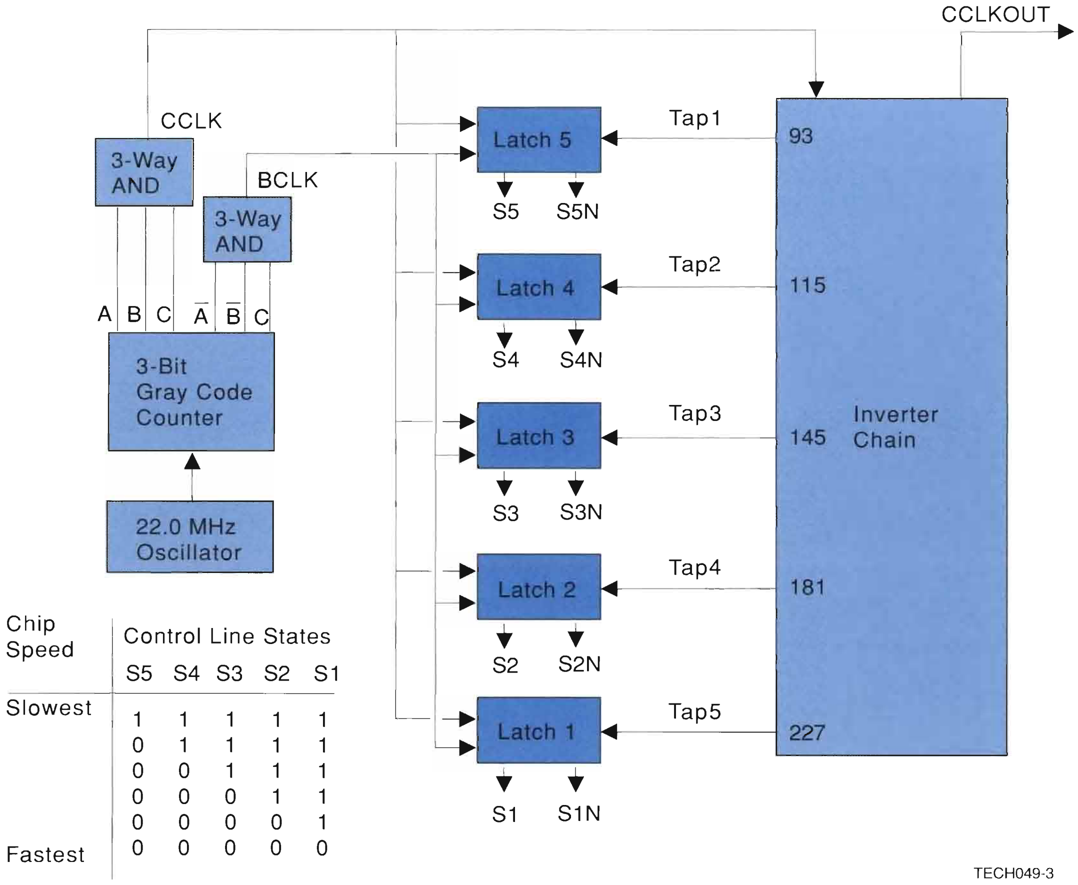
A variety of I/O circuits were available to interface with other chips in the system. Included were receivers, push-pull drivers, common I/O three-state drivers, and common I/O open-drain. Both TTL and CMOS drive levels could be selected. Drivers could be selected with different output impedances to match transmission line characteristics.

AS/400 Digital Slope Control of Off-Chip Drivers

The AS/400 architecture places a high demand on the chip I/O system. The system is characterized by wide data paths to cache, main storage, and control storage. These buses are not only wide, but they are heavily loaded and must be fast to support the high performance of the system. More than 100 drivers must switch at once per chip on a multichip module, with over 100 nets that leave the module switching at once.

Drivers with a totally uncontrolled turn-on rate typically have an 8:1 range in current turn on rate. That is, if at slow process, voltage, and temperature (PVT) conditions, the driver turns on at 10 mA/ns, it will turn on at 80 mA/ns under fast PVT. For a given package, the number of drivers allowed to switch at once is determined by the noise generated at the fast PVT. Performance is determined by the turn-on rate and current at slow PVT. In an ideal case, there would be no variation in turn-on rate. In actuality, however, there is variation, and driver performance always degrades from the noise-limited fast PVT case.

To reduce driver variation with PVT, a family of drivers was designed that dynamically adjusts themselves as temperature and voltage change, as well as for process variations.



Chip Speed	Control Line States S5	S4	S3	S2	S1
Slowest	1	1	1	1	1
	0	1	1	1	1
	0	0	1	1	1
	0	0	0	1	1
	0	0	0	0	1
Fastest	0	0	0	0	0

Figure 78. *Performance Sense Element*

Each AS/400 logic chip receives a reference 22-MHz signal from a crystal oscillator. Circuitry on the chip dynamically compares this frequency against driver performance (see Figure 78). If driver performance is determined to be lagging, additional devices are gated on, increasing current drive and turn-on speed of the driver (see Figure 79 on page 151). Driver performance needs continuous monitoring to guard against temperature changes during operation and low-frequency-supply variations.

The drivers have a finite number of devices that can be gated in. The number of devices imposes a limit to the degree of control the drivers can maintain. In the AS/400 implementation, the current turn-on rate can be controlled to a 2:1 maximum-to-minimum range. A controlled current turn-on rate provides substantial system performance advantages over uncontrolled drivers.

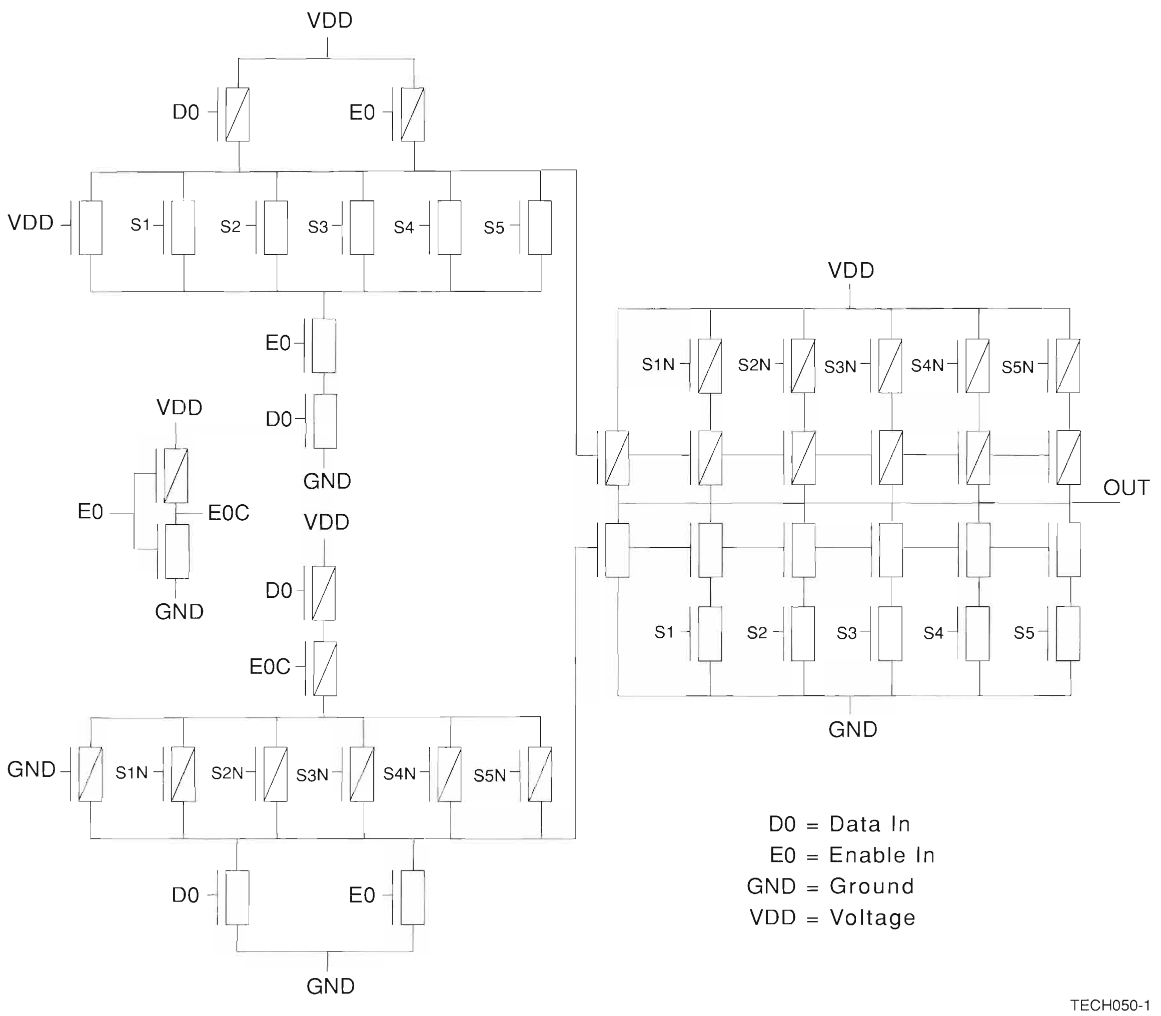


Figure 79. *Driver Schematic*

Most data buses in the AS/400 system are implemented as three-state dotted nodes. One driver receives control of the bus at the same time a different driver relinquishes control. As clock skews are considered, this introduces the problem of contention (both drivers momentarily active and driving in different directions for a brief period). The AS/400 drivers have been designed to go into tri-state quickly while turning on at a controlled rate, greatly reducing or eliminating the contention problem.

AS/400 Packaging Technology

The AS/400 system is designed for great flexibility in range, allowing use of from one to six main storage cards and one or two processor cards.

The processor consists of two multichip multilayer ceramic modules (MCMs). Each module is 44-mm square and supports 545 total pins of which 385 are available for signals. The modules have 12 wiring planes interspersed with power planes, which provide power to the chips and maintain signal line impedance. Electrical characteristics of module wiring are similar to those used on IBM mainframes. The processor card holds two MCMs. This module technology is used to package the chips close together for high performance. The package also provides a high pin count, controlled impedance, and low effective inductance.

One MCM is a processing unit containing three 12.7-mm CMOS logic chips that perform the fixed point, floating point, and logical operations of the processor. The other MCM is a storage control unit containing two 12.7-mm CMOS logic chips and up to two static random-access-memory (SRAM) chips depending on the AS/400 model. This MCM handles data storage to main storage and cache.

The main storage cards contain two logic modules that queue main storage requests and interface between the main storage chips on the card and the processor. These modules are the same as those used on the processor card (44-mm square, 545 total pins).

Although these modules contain only one 12.7-mm CMOS chip, they were required to support the high I/O count of 320 signals from the chip that must leave the module.

AS/400 Clock Generation and Distribution

There is a single oscillator and primary clock module in the system. Up to nine cards in the system run synchronously with the main processor; each has its own secondary clock generation module. Both the primary and secondary modules are bipolar analog chips. Each secondary module receives a clock synchronization signal from the primary clock module. Each secondary module generates four copies of four clocks (each 90° phase-shifted) that are distributed to the required logic chips at the system cycle time (refer to Figure 80).

Clock skew at the card and board level was minimized by balancing the wire lengths and loading of all distributed clock signals. Isolating clock signals from other logic signals also helped reduce clock skew caused by coupled noise. The clock synchronization signals were low-voltage level differential pairs, which helped reduce electromagnetic interference and improve common-mode noise rejection. Much time and effort were spent to ensure proper power supply noise isolation to reduce clock jitter in the phase-locked-loop (PLL) clock distribution.

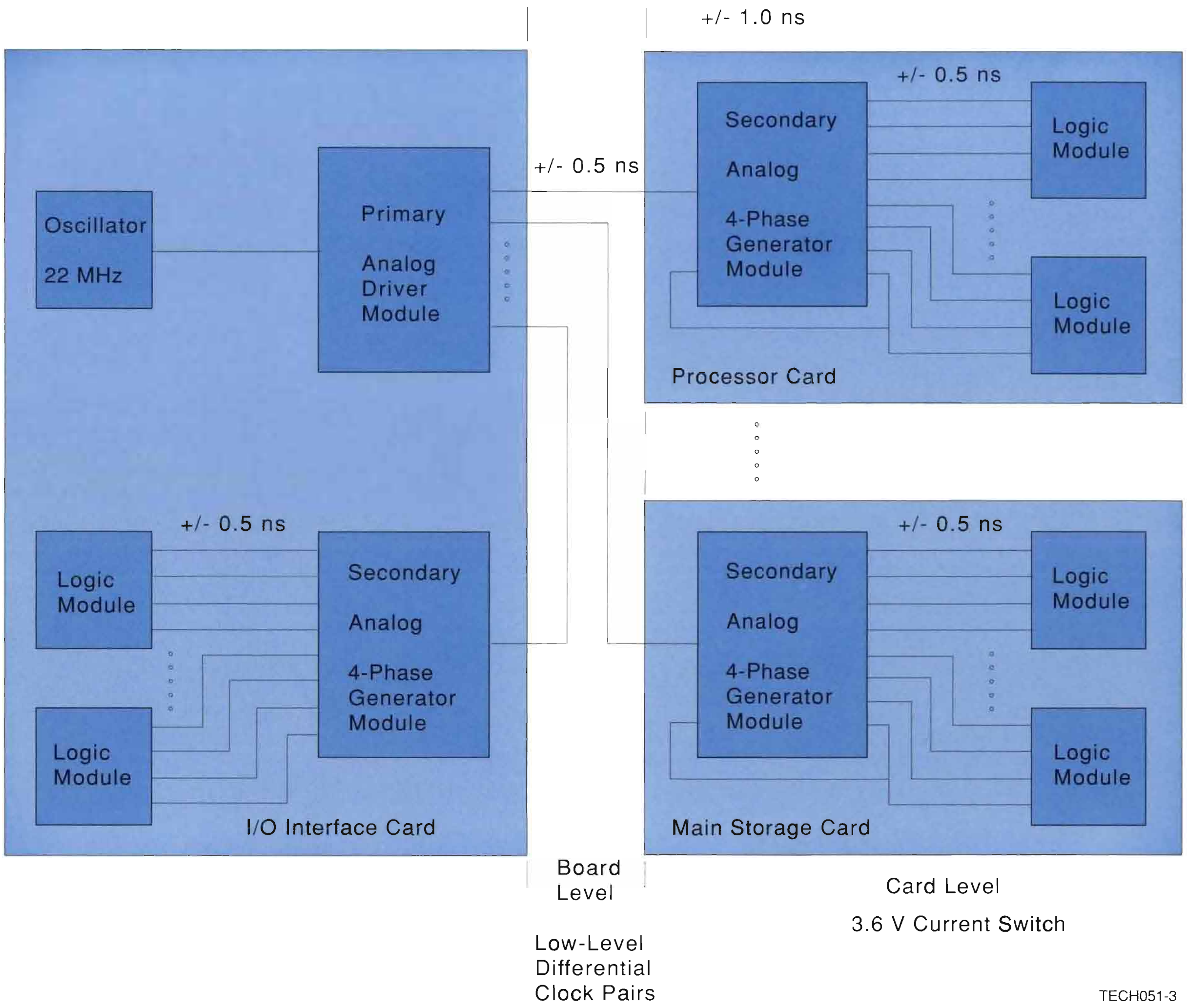


Figure 80. *Clock Generation and Distribution*

The requirement for four-phase clocks at the chip level is a result of a design philosophy that requires mid-cycle array clocks, L1/L2 clock gating, and staggered driver gating for simultaneous switch control.

Due to the application-specific-integrated-circuit (ASIC) design environment, the methodology used to control clock skew is a combined circuit and physical design solution. A key attribute is that this methodology does not affect the chip design turnaround time. It was used on each of the nine logic chips in the system in a turnkey ASIC environment. Chip designers did not have to worry about clock generation and repowering trees. Synthesis programs were developed to automatically insert a circuit-specific version of the clock tree (clock distribution network) into a chip design that had been synthesized from a hardware description language. Software was also developed to automatically balance the clock trees to predefined loading limits. This effectively controlled both the on-chip and chip-chip clock skew.

The circuit portion of the solution consisted of a variable terminator (capacitor) and a variable delay circuit. Use of these circuit concepts within the clocking methodology reduced the stress on the placement and wiring (physical design) tools. Clock nets (interconnections) did not need to be wired to a fixed capacitance; however, there was a range of capacitance within which they had to be wired. This decreased the physical design turnaround time and had less effect on the wiring of remaining critical nets.

The terminator circuit had 20 programmable values of capacitance ranging from 0.05 pf to 1.0 pf in 0.05-pf increments. The terminator circuits were used primarily on the first two levels of the clock tree to help achieve the balanced loading required to reduce clock skew. One or more terminators were used to balance the total net capacitance (sum of the gate and wiring capacitance) to a fixed value. Net terminators were not added to every net in subsequent levels of the clock tree. (This would have required an excessive number of terminator circuits.) For these levels of the clock tree, terminators were only needed when the circuits were not fully loaded (final branches of the tree). The terminators brought the total capacitive load to within the specified range for the given level.

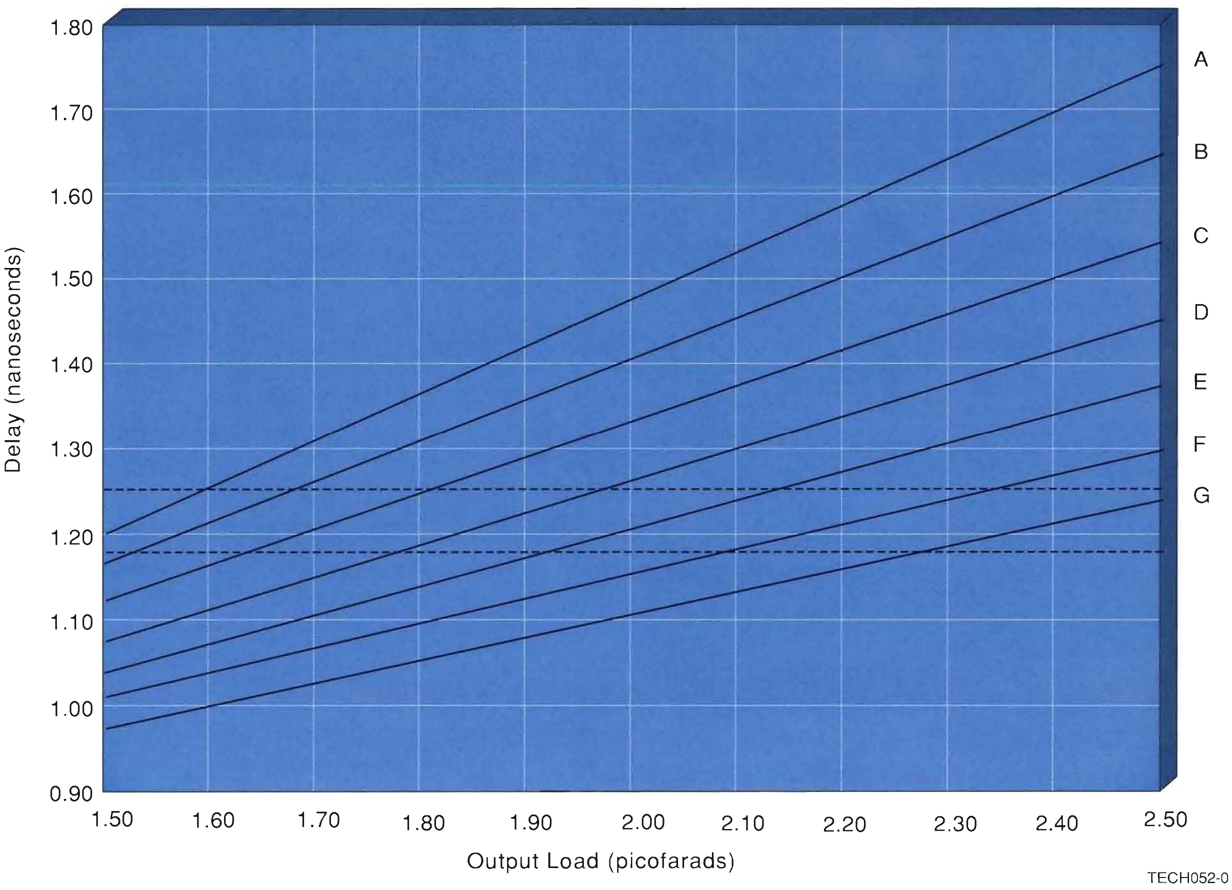


Figure 81. *Variable Delay Circuit*

Special variable delay clock circuits were then used to provide a tightly controlled delay over the specified range of capacitance. The circuit delay characteristics shown in Figure 81 demonstrate a 70-ps delay variation for a 1-pf loading variation. The number of delay levels can be determined by the capacitance range and the desired delay variation associated with it. The more delay levels a circuit has, the more tightly its delay can be controlled. Because every delay level for a particular circuit occupies the same area on the chip and changing delay levels does not alter the chip wiring, the desired delay level of these circuits was selected after the physical design. From a circuit count perspective, the use of the variable delay circuits for clock tree repowering was more efficient than adding terminators to every net.

All net terminators were added prior to entering physical design. A program to set the performance level of the terminators was run after the physical design was complete and after the wiring capacitance necessary to balance the clock nets was known.

A typical clock tree consists of four levels: a receiver level, a clock-generation level, a repowering level, and a final clock-driver-repowering level. These levels are shown in Figure 82. Each level of this clock tree has a unique flag associated with it (CF1-CF4A). Each of the flags or levels has loading guidelines to minimize on-chip and chip-chip clock skew. These loading guidelines are recognized by the clock tree connection programs, the placement and wiring programs, and the post physical design software, which are required to balance the clock tree loading.

Using this methodology resulted in on-chip clock skews of 1.75 ns, chip-chip (same card) clock skews of 3.5 ns, and chip-chip (different card) clock skews of 4.5 ns.

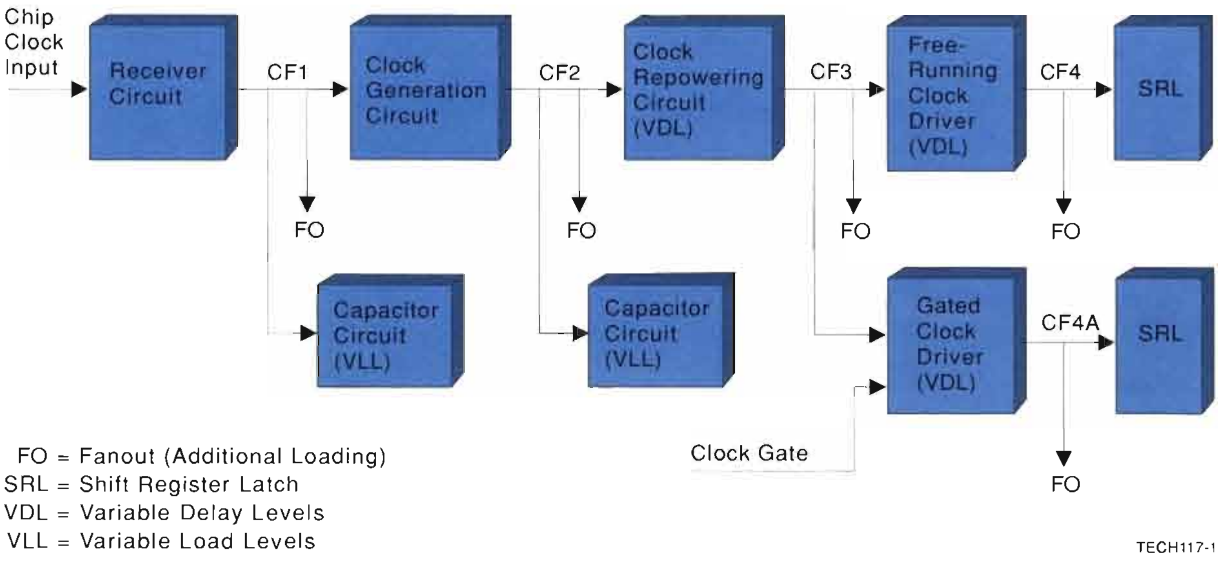


Figure 82. *Clocking Methodology Example*

Conclusion

The emphasis in technology selection for the AS/400 processor was to improve designer productivity and quality. Technology capability was selected to meet system requirements from cost, performance, and productivity perspectives. While the application of the technology did not necessarily result in the ultimate technical capability, it did result in a high quality, high productivity implementation.

Acknowledgments

The authors would like to acknowledge the development efforts of the logic, SRAM, and multichip module packaging teams in East Fishkill, Burlington, and Rochester.

System Processor Self-Test

Describes self-test and its contributions to testing the AS/400 system processor during the initial program load.

Steven M. Douskey and Kerry T. Kaliszewski

Introduction

Self-test can describe anything from a general total test package included in a system to specific algorithms created by a random pattern generator. This article describes the use of built-in self-test techniques to verify internal combinatorial logic and describes deterministic patterns to validate chip-to-chip interconnections in the AS/400 **system processor**. The processor function is not used directly but rather is tested through a scan interface by separate service processor code and hardware.

Overview

Self-test procedures have been added to the functional test strategy in the Version 2 AS/400 processor. These procedures use the service processor to perform various operations on each system processor card during every initial program load (IPL). In a fully configured system with multiple processor and main storage cards and a bus adapter card, over 1,000,000 circuits and almost 3,000 chip interconnections are self-tested.

The **service processor** provides IPL functions, the interface to the control panel, and assistance with failure isolation in the system processor. Self-test code and data are loaded into the service processor random access memory (RAM) during the IPL (see Figure 83). This code runs the service processor hardware through a series of operations, including test data generation, scanning, clocking, compression of resultant data, comparison with expected values, and failure isolation.

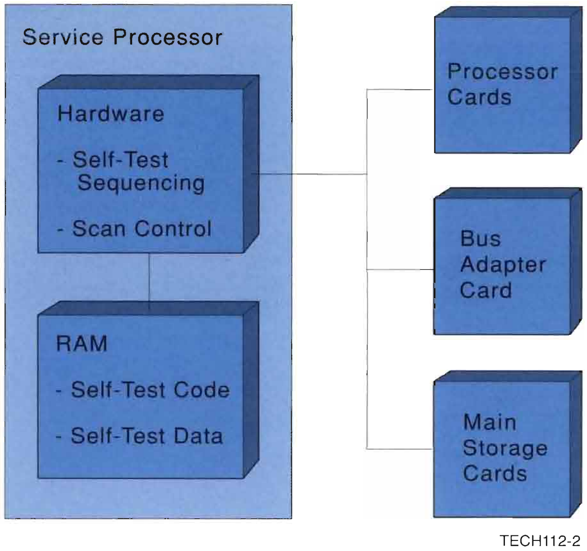


Figure 83. *AS/400 Self-Test Hardware*

During IPL, self-test runs after the service processor basic assurance tests (BATs) verify the self-test control logic on the service processor and after the scan interface BATs verify the self-test interface to the system processor. However, self-test runs on each system processor before that processor performs even the simplest function. In this manner, the tester hardware (which is actually part of the system) is already verified before using it in subsequent tests. Using logic only after it is tested is key to full failure isolation.

Shortly after self-test, diagnostic horizontal licensed internal code runs. It now can add self-test to its base of previously successful tests and adjust its fault diagnostics accordingly. This preserves the IPL test strategy of building a tester from a set of previously verified logic and adding logic only after it too has been tested. This proves to be an effective way to maintain efficient error detection, while maximizing field-replaceable-unit (FRU), module, and even net isolation.

Boundary scan is a necessary part of the technology for self-test. **Boundary scan** isolates the internal logic on each chip from external interfaces. This is accomplished by placing a shift register latch (SRL) on most of the chip input/output (I/O) ports. The notable exceptions to this are clock and scan control I/O. Because the internal logic of each chip can be tested individually without interacting with other chips on the card, precise chip isolation can be accomplished. Furthermore, boundary scan latches surround each I/O pin, simplifying the control of an interconnection test.

The two parts of the Version 2 AS/400 self-test are chip test (or pseudorandom pattern self-test) and wire test (or chip interconnection test). These are described individually in the next two topics.

Chip Test

Chip test uses built-in self-test techniques to check internal logic on a chip (see Figure 84). A pseudorandom pattern generator (PRPG) creates the input test data dynamically. The chip's scan rings are loaded with the PRPG data through scans controlled by the service processor hardware. The data enters the chip at the scan-data-in (SDI) port. Clock control logic then cycles the system clocks (which are used as test clocks during self-test). After the clocks are run, the resultant data is scanned out through the scan-data-out (SDO) port and compressed into a multiple input signature register (MISR).

In the AS/400 implementation of self-test, all of the chips on a card are tested in parallel. After a specified number of test cycles are run, the MISR contains the compressed test results for the card. This result, called a **signature**, is checked against an expected value stored in the service processor RAM. If the signatures match, the next set of test cycles is run. If they do not match, individual chip signatures are gathered and compared against expected values. This is how individual failing chips can be isolated while still maintaining efficiency, parallelism, and speed.

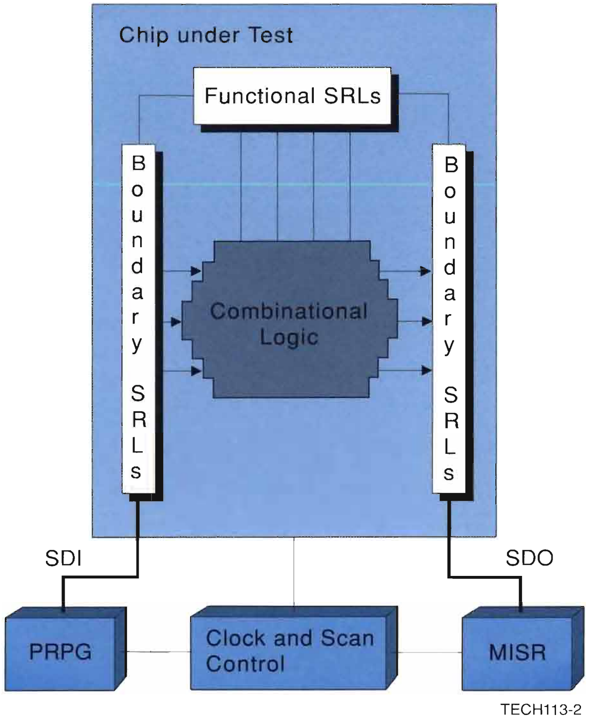


Figure 84. *Chip Test Hardware and a Chip under Test*

Most built-in self-test implementations are purely stuck fault tests. A single test clock pulses after the test data has been set up in the logic for a relatively long time. This clock captures the resultant data into SRLs. The Version 2 AS/400 system processor is designed to include an additional clock, called a release clock, which is pulsed before the capture clock. Pulsing this new clock allows the test data to propagate through the logic before encountering the capture clock. Additionally, both of these clocks are system clocks with the same timing requirements as those run during the functional operation of the machine. This sequence creates a test that directly relates to the timings required within the system. This enhancement has proven to be valuable. It has successfully detected transition faults that would have been missed by other built-in self-test implementations.

Because the hardware generates the input data and compresses the resultant data during chip test, a comprehensive test can be performed without the need to store large amounts of test data. Over 1 billion bytes (one billion equals 10^9) of test data are run through the processor complex every IPL, but only 3,600 bytes must be stored for chip test.

Wire Test

Wire test (or chip interconnection test) is used to verify boundary-scan-to-boundary-scan paths on the Version 2 AS/400 system processor and bus adapter cards. Boundary scan is used extensively to facilitate the application of deterministic test patterns for diagnosing interconnection (net) faults. Stuck drivers, open interconnections, stuck receivers, faulty driver enable circuits, shorts to power, shorts to ground, and shorts between functional nets are all detected by wire test.

Wire test consists of two parts: stuck driver test and shorted net test. The deterministic data (test vectors) used for these tests is generated by a program using simulation models of the chips. This program generates the vectors for the stuck driver test so that every enabled driver in a particular test drives to the same level. This assures that shorted nets do not affect the results of this test. Vectors are also generated so that every driver in each chip is at some point enabled for both a test to a logic level of zero and a test to a logic level of one. Similarly, a program selects shorted net test vectors using the fact that the stuck driver test completed successfully, and focuses on detecting bridging faults. This data is stored in the system for all the release levels of the system processor.

The stuck driver test uses the scan interface to load the test vectors into the boundary scan latches of each chip (see Figure 85). During scanning, the outputs of the chips are disabled by a chip driver inhibit (not by individual driver enable latches). When the scan-in is complete, the outputs are enabled. The receiver clocks for all chips are then pulsed, capturing the data on all the nets into the corresponding receiver latches. The outputs are again disabled, and the data from each ring is scanned out. The output data is then compared against expected results. If a mismatch is found, a system reference code is generated that points to the failing net.

Once the stuck driver test has completed successfully, the shorted net test is run. In the shorted net test, a series of vectors drives each net to a different value than each other net in at least one test. When two nets are shorted and their drivers have opposite values, some of the receivers latch unexpected results. Failing nets are calculated using the expected results and the combinations of failures in the various tests.

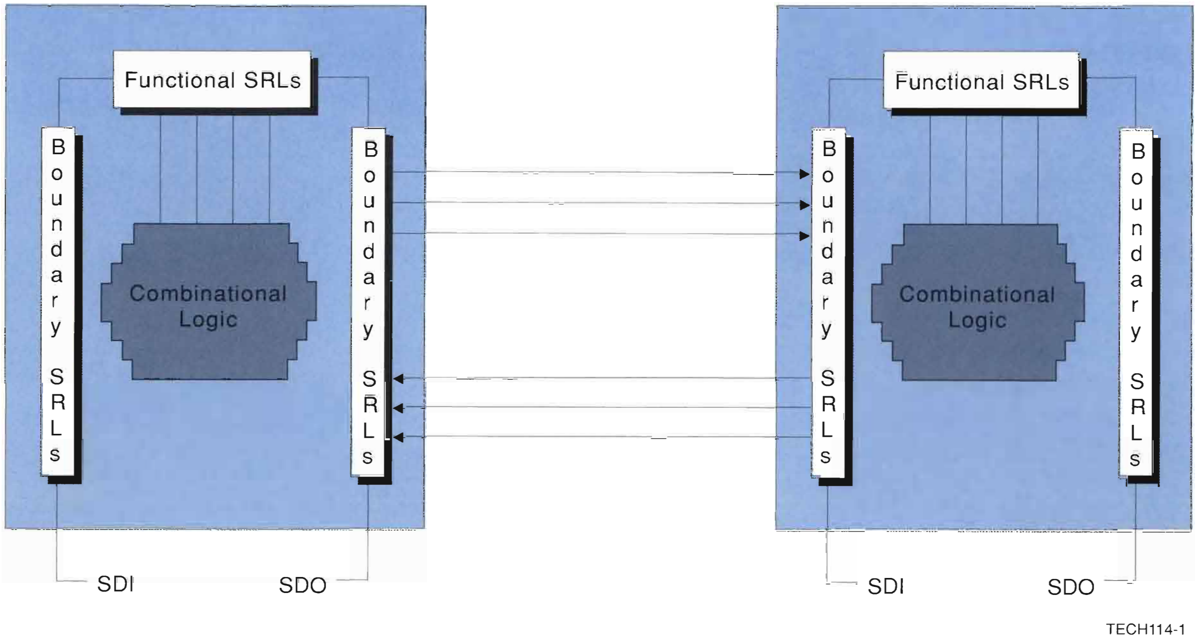


Figure 85. *Boundary-Scan-to-Boundary-Scan Paths Tested by Wire Test*

Both parts of wire test not only verify the hardware during every IPL in the customer's office but are helpful in problem isolation in manufacturing and early system build. They point directly to the defective net or pair of nets.

Conclusion

The self-test procedures are a strong addition to the Version 2 AS/400 test strategy. Chip test not only points directly to the failing chip but is also effective in detecting transition faults. Similarly, wire test significantly improves card defect location, reduces repair time, and reduces cost through accurate and quick net isolation. Additionally, this manufacturing quality test is part of the IPL, making it available in every customer's office.

Acknowledgments

This project was made possible through the support of many individuals. The self-test team would like to recognize the contributions of Mike Weed for wire test definition and implementation, Patricia Graham for early kickoff and organization, and Brian Kozitza and John Elliott for tools interface and support. Thank you all.

References

1. Wagner, P., "Interconnect Testing with Boundary Scan," *IEEE 1987 International Test Conference*.

2. Peterson, W.W. and Weldon, E.J., Jr., "Error-Correcting Codes," *The MIT Press*, Second Edition, Cambridge, MA and London. 1972.

Design Methodology

Describes the architecture and design methodology used in the AS/400 system processor.

Quentin G. Schmierer and
Andrew H. Wottreng[8]

Introduction

The processor architecture design was key to improving the performance of the second generation AS/400 system. Before the design began, it was clear that technology improvements alone would not be enough to achieve the performance objectives. The processor architecture had to be greatly enhanced. The major architectural additions were a store-through cache, expanded main storage addressability, improved floating-point performance, and the ability to operate in a multiple processor environment. The multiprocessor capability had to be implemented with minimal effect on the operating system. To make the design project successful required design tool enhancements and increased reliance on logic synthesis. The remainder of this article briefly describes the processor architecture and the design methodology used to create the processor.

Architecture

The AS/400 processor is fabricated from an IBM low-power, standard cell complementary metal-oxide semiconductor (CMOS) technology. There are five 12.7-mm logic chips and two 8,192 x 18 static random-access-memory (SRAM) chips packaged on two 44-mm multichip multilayer ceramic modules (MCMs). The cache and control storage are individually packaged SRAMs. The processor clock cycle is 45 ns, worst case. This was the same clock cycle as the first generation processor, using a less dense bipolar technology.

The processor fetches and processes internal microprogrammed interface (IMPI) instructions. As the name implies, each IMPI instruction is actually processed as a microprogram. An IMPI instruction may be 2 to 6 bytes in length, aligned on byte boundaries. The IMPI instruction is processed as a sequence of one or more horizontal licensed internal code (HLIC) words. Each HLIC word is 56 bits in width. This long width allows several operations to be processed independently during the same processor cycle. The control storage array contains the most commonly processed HLIC words. Additional HLIC is loaded from main storage in an overlay area as required during operation. Figure 86 on page 161 shows the basic components of the AS/400 processor architecture. The main storage interface is shared with the I/O subsystem.

The fixed-point unit operates under the direction of control words that use fixed-point arithmetic and other logical operations. It contains an array of general purpose registers (GPRs) accessible only to HLIC. A 32-bit arithmetic logic unit (ALU), a halfword decimal ALU, and

[8] © 1991 IEEE. Reprinted, with permission, from *Proceedings of IEEE International Conference on Computer Design*; "IBM AS/400 Processor Design Methodology"; Boston, MA, October 14 through 16, 1991; 440–443.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the IEEE copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Institute of Electrical and Electronics Engineers. To copy otherwise, or to republish, requires a fee and specific permission.

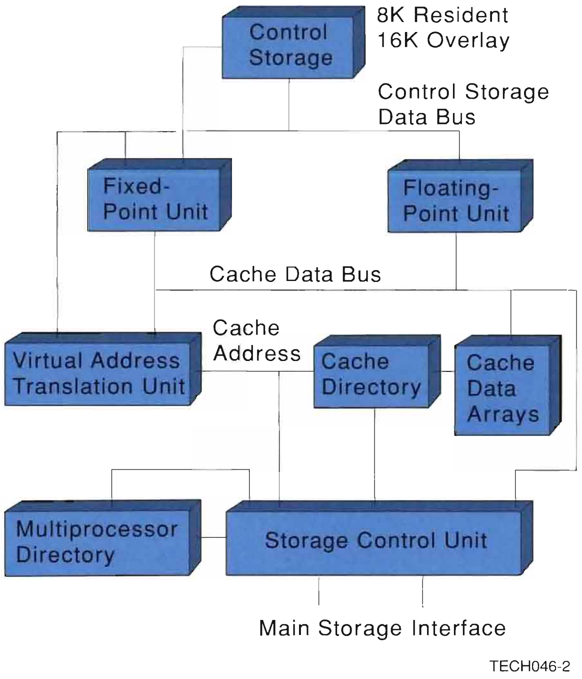


Figure 86. *AS/400 Processor*

a halfword multiplier are also implemented. The logic required to fetch the next HLIC word, based on the current processor state, is in this unit. The first HLIC word for each IMPI instruction is based on the IMPI operation code. A floating-point unit, adhering to the Institute of Electrical and Electronics Engineers (IEEE) floating-point standard [1] is included. It contains eight floating-point registers (FPRs) capable of single- and double-precision arithmetic on addition, subtraction, multiplication, division, comparison, and square root.

All references to storage in an IMPI instruction are through virtual addresses. The virtual address translation (VAT) unit converts virtual addresses into real main storage addresses. The operating system works with 16-byte pointers, which support 8-byte virtual addresses, although only 6-byte addresses are directly manipulated in the hardware. The most recently used virtual addresses are kept in a translation look-aside buffer. Fetch and store operations to main storage are initiated, as required, by the HLIC words. The IMPI instructions directly manipulate sixteen 48-bit registers. They may contain addresses or program data. A set of HLIC accessible registers called resolved address registers (RARs) store translated virtual addresses for reuse during IMPI instruction processing. Hardware in the fixed-point unit handles IMPI instruction fetching and buffering.

The storage control unit (SCU) manages accesses to main storage. Data may be retrieved from either the 128KB (KB equals 1,024 bytes) cache or directly from main storage. The cache contains both IMPI instructions and data used by the IMPI instructions. For both instruction and data accesses, the SCU detects cache misses. Cache misses occur when accesses of main storage locations do not find a copy of the main storage data in the cache. When a cache miss occurs during a fetch, the SCU fetches a 32-byte integrally aligned block of data from main storage and stores this data in the cache. This 32-byte block is called a cache line. The SCU also handles store operations. Because the cache is store-through, all store operations are done to both the cache and main storage. The SCU's secondary functions are resolving translation look-aside buffer misses and arbitrating the use of the main storage interface with the input/output (I/O) subsystem. Multiprocessor contention for the main storage interface is also arbitrated here.

The main storage interface consists of a command bus and a 64-bit wide data bus. The I/O subsystem can store and fetch data directly from main storage. Up to eight main storage cards, each containing as much as 64MB (MB equals 1,048,576 bytes), may be attached to the interface. Other processors may also be connected. When a multiprocessor configuration is implemented, the processors share a single copy of main storage. All interface buses are synchronous with the processor clock using sophisticated clock synchronization logic. Error correction, diagnostics, and dynamic random-access-memory (DRAM) refresh are implemented on the main storage cards in two 12.7-mm CMOS logic chips. Multiple I/O buses are connected to the processor. The first bus is wired directly, while the remainder are connected using fiber optic cables. Redundancy paths are implemented to provide an alternative means for data transfer in case one of the fiber optic channels fails. Each I/O bus is 32 bits in width and asynchronous. (For more information, see the article "Input/Output Bus Using Fiber Optics" on page 172.)

Design Methodology

The design methodology for the AS/400 processor involved significant change. Designers described a new architecture and converted that description into working hardware as efficiently as possible while meeting schedules. The designers chose, for the first time in a processor design, to use logic synthesis for the transcription process. They selected the very high speed integrated circuit (VHSIC) hardware description language, better known as VHDL, as a design description language. Many designers had experience with earlier internal register transfer languages. Some of the designs had been synthesized into logic, but success had been mixed. Synthesis seemed to work well as long as performance was not a major issue. The earlier register transfer language did not allow much control over the synthesis process. VHDL, with its mix of functional description constructs and attribute information, offered a way of putting more designer input into the synthesis process. This ability was critical in achieving the design goals.

Figure 87 shows the general flow of the design process. This article explains the process steps and how they were changed from previous projects. One of the first differences in the process was documentation. Because of the massive amount of change involved, documentation was very important. Without accurate documentation at the start, subtle errors could creep in and escape detection until late in the schedule. Designers spent much time preparing a formal architectural specification. This was reviewed by each design team and used as a starting point and reference for each design. The HLIC format was also defined at this time.

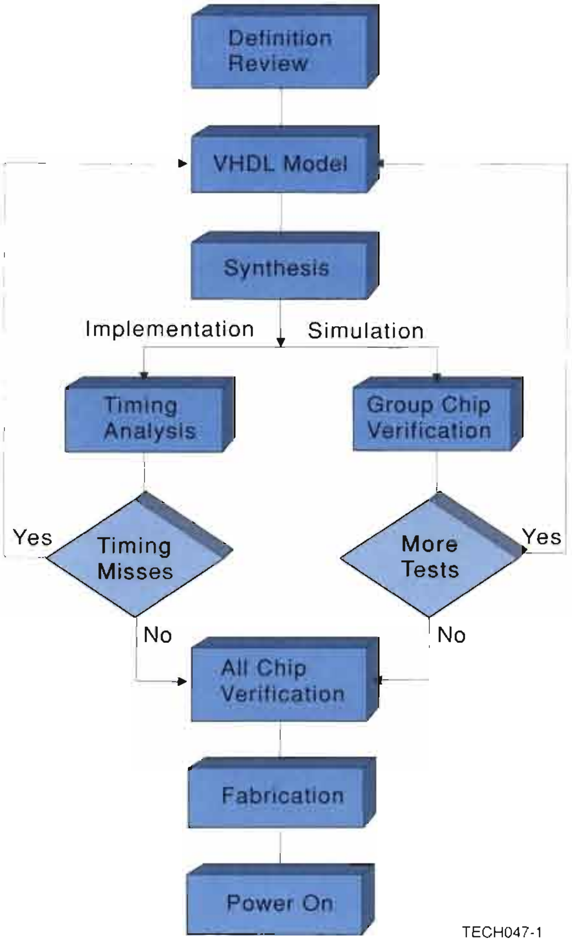


Figure 87. *AS/400 Design Process*

The chip design teams began the implementation phase by defining chip partitions. The processor was targeted to fit into five CMOS chips. Once the function of each chip was roughly defined, an I/O specification document was created. It contained a detailed description of each chip level signal and how it interconnected. The design teams reviewed the document periodically and worked out differences and vague definitions. A program screened the document for changes and alerted the proper chip teams. All changes had to be reviewed and approved. This reduced errors and misinterpretations significantly and led to a better first implementation. The chip interfaces were defined as VHDL entities. Each chip was internally divided into partitions. Partition interfaces were also defined as VHDL entities and component instantiations within a hierarchy.

In addition to the formal documents, chip teams attempted to document their design as they wrote it. In some cases, this was in the form of a separate chip workbook. In most cases, the workbook was simply a collection of commented VHDL models. The comments were written first to describe the logic function; the actual VHDL defining the function was added later. The VHDL functional description was not, by itself, sufficiently clear to document the design for review purposes. Even more clarity was sacrificed as the design was modified later on to resolve logic timing problems.

Logic Entry

The chip logic designs were described in VHDL. To allow reuse of existing synthesis and simulation tools, some compromises were required. The major compromise was to use a subset of the language. VHDL allows both sequential and concurrent statements to describe logic function. Only the concurrent set of VHDL statements, excluding process statements, was used. The subset correlated closely in function to previously used design languages. This shortened the designers' learning curve of the language as well. VHDL attribute information was heavily used to provide synthesis directives. The designer was able to influence the decisions made during synthesis by this means. The attributes consisted of hints and commands. Hints were simply suggested solutions that should be attempted first. Commands were explicit directions that forced a particular solution. These attributes made up for weaknesses in the synthesis algorithms and gave the designer pinpoint control of results. Control of results was important in achieving a high performance design. Synthesis produces acceptable implementations in 95% of the design, but the last 5% requires precise designer control.

Standard text editors were used to enter the VHDL. Textual comments and data flow diagrams were freely mixed with the VHDL description. This was valuable later when the VHDL was modified. Invariably, modifications made the VHDL less readable. Partition sizes varied from 200 to 5,000 lines of VHDL. The relationship between lines of VHDL and actual logic cell count were inverse. Control and sequential logic required many lines of VHDL, but synthesized into small amounts of logic. Data flow could be compactly described with vectored signals and simple operators in a few lines of VHDL but synthesized into large amounts of logic.

Because designers depended on text entry for the VHDL descriptions, anything that simplified the task was welcomed. There were many logic constructs where the VHDL description consisted of declarations, attributes, and concurrent assignments. The information was scattered throughout the model to meet the requirements of the language syntax. It was time-consuming and difficult to maintain such constructs in a large model.

To ease the entry task, a VHDL macro preprocessor was developed. The preprocessor keyed on VHDL macrostatements within a description. The macrostatements were expanded into normal VHDL statements and placed in the proper areas of the VHDL model. Attributes were entered as meaningful keywords on the macrostatements. All of the information for a particular construct was contained on the macro statement. Many of the more cumbersome structural attributes were generated automatically along with tedious logic connections. For example, test generation scan rings were connected automatically by the preprocessor. As a result of this technique, the designer entered only about one-half the actual VHDL code required to describe the design; the remainder was generated by the macro preprocessor.

The VHDL attributes were important in customizing the VHDL description to the technology. The selection of receivers, drivers, and I/O books was very critical to the processor design. It was difficult to write synthesis algorithms that selected the proper I/O books. Attributes that specified exact book types and power characteristics were a better solution. VHDL component instantiations described each I/O book. Attributes attached to the component specified characteristics that could vary from one instance to the next. Some drivers had multiple output pins, each of which was designed to drive a specific output impedance. The choice of driver pins was done with an attribute. The physical location of the I/O books and the power levels were also specified this way.

Logic Synthesis

Logic synthesis is an algorithmic process for taking a logic description written in a high-level form and transforming it into a gate-level description that can be manufactured. The VHDL description for each chip was synthesized into logic using the logic transformation system (LTS). LTS is an internally developed logic synthesis system designers have used since 1985. The VHDL was synthesized for both simulation and fabrication; however, the synthesis differed for each. For simulation purposes, a path that translated the VHDL into technology-independent logic gates was used. Care was taken to correlate the logic description with the VHDL. The technology-independent logic gates were optimized for an existing event driven simulator. Little logic optimization was performed. VHDL hierarchy was maintained. The desired result was a gate-level description that closely matched the original VHDL description. It could be easily correlated to the VHDL model statements as well. This correlation allowed the designer to verify a VHDL model without the benefit of a VHDL simulator. VHDL allowed this to be accomplished.

The synthesis path for fabrication was much more complex. Designers chose to synthesize the chip logic with the internal VHDL hierarchy removed. This was a case of expediency winning over good judgement. The synthesis system had handled complete chip descriptions up to this point, and it was a lower resource cost to continue this way. Capacity projections indicated it was still possible with the target technology. Designers accomplished the task in this manner, but it cost in long synthesis times and loss of synthesis control. Future projects will use hierarchical synthesis. The synthesis system has been enhanced to support hierarchy. Early results with the new methodology have produced better synthesis results and quicker iterations.

Synthesis has always been a double-edged sword. On the one hand, it frees a designer from the mundane and error-prone task of manually converting a logic description into a hardware implementation. On the other hand, it hides the implementation from the designer and makes it difficult to see where an implementation is not satisfying requirements. A logic analysis tool that displays logic paths and timing information is essential for analyzing problems. The logic must be annotated as much as possible from the original VHDL model to help trace through it. We had several such tools that proved valuable.

Designers experimented with numerous methods for influencing the synthesis tools. VHDL attributes were useful for this. Designers used them at the chip boundary to define input arrival times and expected output times. The times were converted by synthesis into implementation goals. They also used attributes internally to control special logic paths. A common situation was a path that appeared to be critical to synthesis but really was not. The path was marked as a noncritical path. Synthesis would ignore such paths and concentrate on more critical ones.

Timing misses often required rewriting the original VHDL description. Commonly this meant writing low-level expressions for the logic. Because much care was taken in writing the description, it was important to preserve the work. Designers used the no-modification and map attributes to control this.

A no-modification attribute attached to a VHDL signal preserved that logical point through to final logic. This attribute prevented synthesis from moving logic forward or backward through the point. The map attribute was even more stringent. It forced synthesis to convert a logic expression attached to it into a technology gate implementation early in the process. This preserved the implementation through to the final logic. Both of these attributes were extensively used to achieve final timing closure.

VHDL was a good choice for logic synthesis. It allowed the designer to describe the logic at either abstract or detailed levels. The lowest levels of detail reached were VHDL functions and component instantiations; both could be written so that they mapped exactly into a technology book. Designers used this technique extensively for arrays and complex books that were present in the technology. It was easier to write this type of description for complex books than produce them through special synthesis algorithms.

Simulation

Simulation of the AS/400 processor was successful, but involved several limitations. The first limitation was the simulator itself. As early users of VHDL, the designers did not have a working VHDL simulator available. As a result, they were forced to reuse an existing gate-level, event-driven logic simulator. It went through three revisions before the project ended.

The simulator the designers used accepted gate-level descriptions as input. It also had its own behavioral language and test case control language. They had to convert the VHDL models into a form called basic design language for structure (BDL/S). It is basically a logic net list. They developed a set of technology-independent BDL/S blocks, simulation behaviors, and synthesis algorithms as part of the methodology. All VHDL simulation models were synthesized into BDL/S.

To make this strategy work well, they had to make it easy for the designer to correlate the VHDL model to the BDL/S description. They had to extract as much information as possible from the VHDL and attach it to the BDL/S blocks and signals. This was done by again making extensive use of VHDL attributes.

VHDL signal names were used to generate BDL/S signal names. The primary means of debugging a test case was with a program called scope. It displays waveforms for nets specified by the designer. The names chosen for nets matched the names used in the VHDL. The hierarchy present in the VHDL was maintained in the simulation models. The logical function that existed in the BDL/S closely matched the VHDL model. With this strong correlation, it was never necessary to look at the BDL/S format explicitly.

The method worked well and is still in use. No designer has ever looked at a technology-independent BDL/S schematic. The ability to annotate the synthesized description from the VHDL allowed designers to reuse an existing simulator with minimum investment. They also found that the same annotation was useful in analyzing timing problems when synthesizing to a technology.

The design was simulated in several distinct environments. These included a single chip environment, multiple chip groups, and, finally, a whole system. Each environment consisted of VHDL hardware models and simulation behaviors that emulated interfaces and arrays, such as main storage, cache, and control storage. Test cases consisted of HLIC instruction streams. These had first been run on a behavioral-level HLIC simulation model. The results of the HLIC model simulation and the hardware model simulation were compared. Discrepancies were either fixed or explained. Some self-checking manual test cases were also used. The system-level simulation was done with a hardware simulation accelerator. A model that encompassed most of the system was used along with system-level HLIC. A portion of the initial power-on sequence and many of the diagnostic functions required later in debugging hardware were simulated here.

Each design group had to generate chip-level test patterns for chip manufacturing. They also were responsible for fixing timing problems. These tasks were done outside of simulation with level sensitive scan design (LSSD) test generation programs and static timing programs. The results were analyzed, and, if changes were required, the changes were made at the VHDL level and resynthesized. VHDL attributes were used to supply the proper test and timing flags required by the programs.

Conclusion

The chip design methodology used on the second-generation AS/400 processor was a mixture of new and old. It was the first processor design to successfully use logic synthesis and the VHDL design language. Designers were able to use the functions of VHDL to mesh nicely with existing design tools. Every new tool required extensive debugging and enhancement before it was productive. Old tools usually required modifications, but they became operational much quicker. VHDL was a more difficult design language to use, but the leverage it provided was worth the extra effort.

Logic synthesis gave designers improved productivity by getting the design into working logic easier. They were able to get into simulation faster and implement design changes more easily. This was especially true in the area of finite-state machine logic. Synthesis could not do the job completely. It is important to be able to guide synthesis toward an acceptable solution. VHDL helped do this productively. Using a combination of synthesis and designer directives, designers achieved an implementation as good as a manual design. Furthermore, because the design is not locked into a specific technology, the same, or a slightly modified design, can be redone in a new technology.

Logic synthesis and VHDL gave the AS/400 design methodology the flexibility it needed to make the project successful. Further enhancements to the process will allow IBM to bring products to market faster and with fewer errors than before.

Reference

1. *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Standard 754-1985, IEEE, Inc., New York. August, 1985.

VLSI Design Automation

Describes advancements to the design automation process for the AS/400 system.

Robert F. Lembach, John M. Borkenhagen, John R. Elliott, and Randall A. Schmidt[9]

Introduction

The design automation process for the AS/400 system is driven by the most advanced very large scale integration (VLSI) technologies available. An automated design process is essential to obtain functional hardware on the first attempt. Recent process advancements described in this article cover the areas of high-level language and synthesis, timing verification, chip physical design, and system test. The process continues to evolve to meet the dual challenges of shorter design time and improved solution quality.

Logic Synthesis

The system processor chip set is described in a hardware-description language (HDL), a specially adapted computer language that describes the workings of digital computers. The particular language used is the very high speed integrated circuits (VHSIC) HDL, VHDL. **Logic synthesis** is the process of converting VHDL to a lower-level language that describes logic structures on an integrated circuit chip for one technology. The output from logic synthesis is a **net list**, which describes the lower-level logic structures.

VHDL is synthesized to net lists using an IBM logic synthesis tool called the logic transformation system (LTS) [1]. VHDL and LTS enhance designer productivity by allowing a high-level technology-independent description, which is subject to many functional changes during the design cycle, to be quickly converted and optimized to accurate net lists. Two synthesis scenarios are used. The technology-independent simulation path builds hierarchical simulation models, maintains the logic structure as it is described in the VHDL, and has fast running time. The technology-dependent path optimizes the area and performance.

The IBM VHDL design system [2] is used to develop the design description. Macrostatements help reduce the complexity of coding VHDL. A simple macro may expand into many lines of VHDL and is used to simplify design entry by automatically generating logic, such as internal scan path and boundary-scan connections. Additional design information is provided to synthesis by using VHDL attributes that can be attached to objects in the model, such as pins, nets, gates, or the model itself [3]. These attributes include flag data, technology data, synthesis directives, and performance criteria for synthesis optimization. The designer may choose to add technology-dependent information to the VHDL description to help synthesis produce the desired implementation at critical points in the logic.

The design system can synthesize hierarchical VHDL models that are collapsed and flattened, or it can stitch together partitions that are synthesized individually. Technology-dependent synthesis is performed on flattened VHDL models to permit logic optimization transforms to work freely across the original partition

[9] © 1991 IEEE. Reprinted, with permission, from *Proceedings of IEEE International Conference on Computer Design*; "VLSI Design Automation for the AS/400"; Boston, MA, October 14 through 16, 1991; 444–447.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the IEEE copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Institute of Electrical and Electronics Engineers. To copy otherwise, or to republish, requires a fee and specific permission.

boundaries, and to eliminate the requirement of providing VHDL timing attributes at partition inputs and outputs. LTS expression processing converts the flattened VHDL model into latches and primitive logic blocks. This logic is optimized by technology-independent transforms to remove redundancies and combine common logic. The logic is then assigned to the target technology, and further optimizations are made to minimize the area and to create a net list that is valid for the technology.

One goal of LTS is to achieve the performance targets specified by the designer as VHDL attributes. LTS has a built-in delay calculator and path analyzer for doing both global and incremental static timing analyses that support multiple clocks, multicycle clocks, and overlapped clocking strategies. LTS uses worst-case delay calculations during performance optimization to improve logic paths that do not meet specified timing requirements. LTS uses best-case delay calculations to pad short paths, which prevents problems caused by overlapped clocks. A comprehensive report, prepared at the conclusion of the synthesis run, assists the designer in understanding the performance of the synthesized logic. Finally, LTS creates a file containing information needed by the statistical timing tool and needed by the timing-driven placement and wiring programs.

Timing Verification

After completion of the logic description, the next step is chip- and system-level timing verification. This process involves performance verification of the system processor and main storage. As shown in Figure 88, chip-level timing verification is performed after completing VHDL definition and synthesis. System-level timing verification is performed after completing chip-level timing and definition of the card logic structure. After completion of chip-level placement and wiring, final system-level timing results are obtained. As changes are made to the design, timing verification is run again to ensure timing requirements are still satisfied. Chip-level timing verification is based on the IBM early timing estimator (ETE) [4]. Statistical support was added to ETE to account for the on-chip process variation of large complementary metal-oxide semiconductor (CMOS) chips in the system design. To pass timing information to ETE, timing attributes are inserted in the chip-level VHDL description and used by logic synthesis. In this manner, logic signals can be ignored or have their timing requirements adjusted as needed. Comprehensive timing reports allow designers to implement logic changes in VHDL. These same timing reports are used to create constraints during chip physical design.

System-level timing verification is based on the IBM arrival time program (ATP). The scope of ATP uses chip-level timing information and system-level delays to generate input arrival

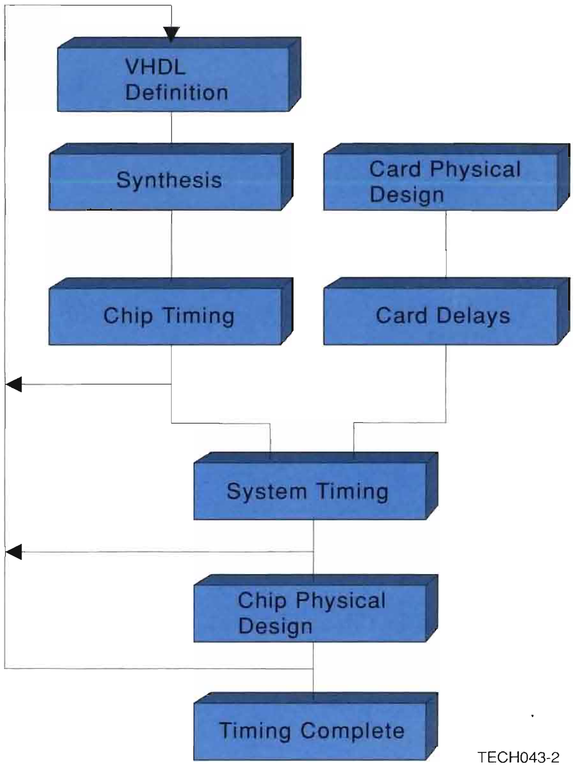


Figure 88. *AS/400 Timing Process*

times for each chip. ATP accounts for contention on bidirectional buses, performs timing tests at logic arrays, supports statistical timing, and creates a report to analyze the simultaneous switching of off-chip drivers. Delay from the driver input of one chip to the receiver input of another chip is calculated using transistor-level models to calculate driver delay. Included are the loading and transmission line effects at chip, module, card, and board package levels for every net. ATP calculates setup, hold, and pulse-width timing tests for static random-access-memory (SRAM) and dynamic random-access-memory (DRAM) chips. ATP propagates delays through SRAMs and DRAMs with delay models. ATP timing calculations are done statistically with the same equations used by ETE.

Bidirectional buses are required to manage the large bus widths and fixed numbers of chip input/output (I/O) pins. Aggressive performance requirements do not allow inactive cycles when changing direction on bidirectional buses. Chip-to-chip skews on driver control logic introduces the risk of driver contention on these buses. ATP calculates the delay effect due to driver contention and adjusts the appropriate arrival times at chip receiver inputs. In addition, the contention duration is calculated and used to assess driver reliability. The contention calculated for reliability purposes is greater than the contention calculated for delay adjustment when a driver is enabled before its data inputs are stable.

The output of ATP is used to analyze noise induced by simultaneously switching drivers. ATP determines the window of switching activity for drivers. Switch activity arises from both driver input changes and contention. Technology information provides the magnitude of noise generated by a given type of driver and the maximum allowable noise for chip and module packages. This analysis allows each chip design to achieve maximum switching activity within the technology limits.

Chip Physical Design Process

The chip physical design process, shown in Figure 89, is cast as a sequence of batch jobs having quantitative measurements for each process step. Recently, steps were added to aid timing closure, manage logically equivalent signals, such as clocks, and handle incremental logic changes. With a globally focused and a highly automated process, designers can synthesize chip designs from VHDL and still maintain high confidence of satisfying chip performance requirements.

Before physical design, a designer runs chip-level timing using wire length estimates based on historical data. These timing results are used to create wire length constraints for the most critical nets that the circuit placement program attempts to meet. After placement, the logic is modified to improve performance and aid in the wiring step. These modifications include clock redistribution, changes to logic repowering, and scan string optimization. After these changes, chip-level timing is run a second time using wire length estimates based on the locations of the pins on the placed circuits. These timing results provide an early indication of final timing results and are used to create wire length constraints for routing. After routing is complete, chip-level timing is run a third time to verify chip performance. Programs compare the timing results of the runs made before and after wiring to identify critical nets that could be optimized further and to determine the expected improvement if the changes were implemented.

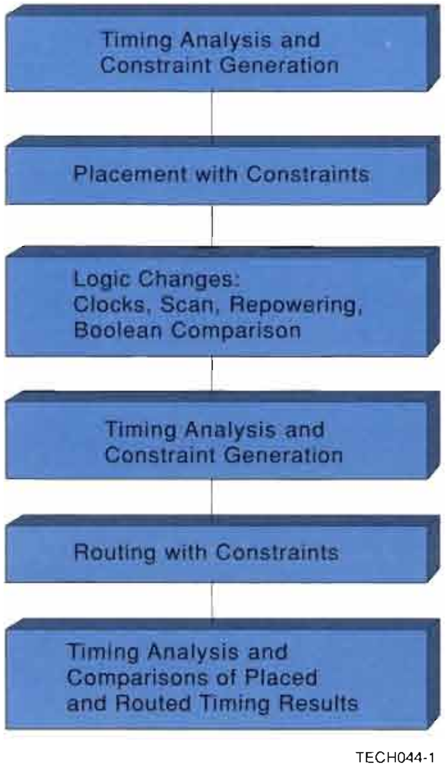

Timing Analysis and Constraint Generation
Placement with Constraints
Logic Changes: Clocks, Scan, Repowering, Boolean Comparison
Timing Analysis and Constraint Generation
Routing with Constraints
Timing Analysis and Comparisons of Placed and Routed Timing Results
TECH044-1

Figure 89. *Chip Physical Design Process*

Clock distribution logic and other logical equivalencies are automatically extracted from the net list. These signals can be ignored during placement and reconnected later using a batch clock optimization program. This program calculates gate and wire loadings and can modify fanout and circuit locations to balance nets or optimize for shortest wire length. Scan strings are rechained to reduce wire length. These logic changes are added to the net list before routing, and a Boolean comparison is run to verify the changes.

Logic changes affecting the physical design are processed incrementally. The goal is to minimize perturbation of the existing placement and routing results. A placement program, based on simulated annealing [5], is used for both initial circuit placement and subsequent incremental placements. To place new circuits, a modified annealing schedule and different quality metrics are used. New circuits are usually placed in open areas while holding existing circuits fixed. Clock logic or scan string changes are also processed incrementally.

System Test

The system incorporates various self-tests that run at initial program load (IPL) to diagnose failures down to individual modules and nets. Using this method eliminates the need for a stuck fault test of the cards. To achieve a high level of confidence that self-testing could replace the traditional stuck fault testing, the tools supporting the generation of the self-tests are exercised to their fullest. Patterns are generated from several software models using generation models. Programs compare patterns created from each model for agreement. All patterns had to be the same for all test cases.

The hardware contains built-in checkers, such as parity checkers, to detect failures. IPL tests exercise the hardware functionality and the checkers to ensure they can be both set and cleared. This is commonly referred to as functional testing. This was extended to include a class of testing known as boundary-scan testing and pseudorandom pattern testing. For boundary-scan testing, the I/O circuits of the chip technology incorporate level sensitive scan design (LSSD) latches. A diagram of the boundary latches within the chip is shown in Figure 90.

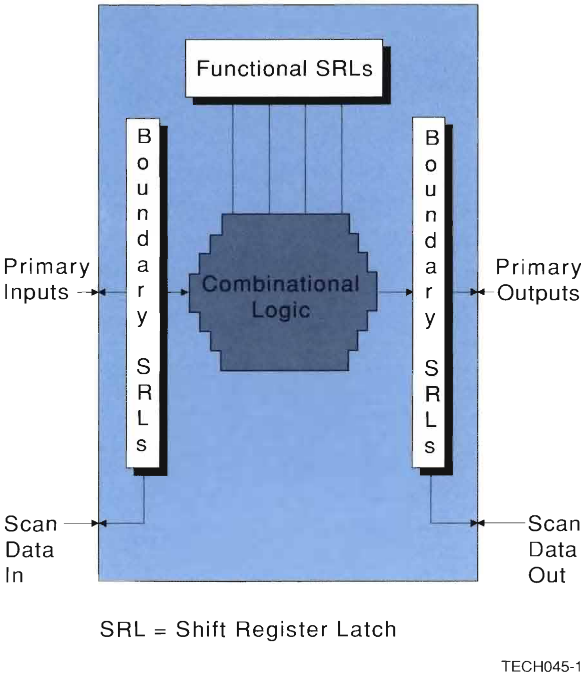


Figure 90. *Boundary-Scan Configuration*

Boundary-scan latches are part of the LSSD scan rings, which are connected serially into scan paths within the chip. All primary input and output pins can be isolated from the internal logic of the chip using the scan latches. A boundary-scan latch on a chip *primary input* provides observation when testing the connection from other chips, and provides control when testing the internal chip logic. A boundary-scan latch on a chip *primary output* provides control when testing the connection to other chips, and provides observation when testing internal chip logic.

Pseudorandom pattern testing replaces the traditional LSSD testing, which relies on stuck fault patterns applied and measured by a tester. To accomplish this, the source pattern generation logic, response data compression hardware, and diagnostic hardware and software are built into the system. Source pattern generation uses a pseudorandom pattern generator (PRPG); the data compressor is a multiple input shift register (MISR). The design system supports random pattern self-testing by generating the patterns with software models, applying these patterns to the hardware models, then collecting the signatures. The patterns can be generated for various clock cycling techniques to achieve maximum coverage of the logic. The signatures are then stored in the code of the machine, where the PRPG is run, using the same cycles and clock sequences as simulated previously to produce signatures collected in MISRs. The signatures are compared during IPL time. If a signature does not match, the PRPG and MISR can be reconfigured to run additional tests to further isolate the failure.

Another class of self-testing is to test the interconnections between modules and chips to ensure continuity of nets. Extending the LSSD design rules so that boundary scan is incorporated on all the chips within the system, the design system assisted in producing a software model to simulate the stimuli and responses of running tests between chips. The sequences of stimuli and responses are stored in the IPL code and run on the hardware to test the chip-to-chip connections for shorts and opens. The greatest benefit of these tests and the previous random pattern test is that they run close to machine speeds.

The AS/400 test process combines random pattern testing, functional speed testing, boundary scan, and failure diagnosis to produce an effective system-driven test. Without the design tools for simulation and test generation of the models, these tests would be less effective.

Conclusion

The VLSI design automation process for the AS/400 system, driven by the latest technologies available, continues to evolve to meet the overall system requirements for performance and product quality. Recent enhancements to the process include high-level language and synthesis, chip and system timing analysis and verification, chip physical design, and system test. Future efforts will focus on continued reductions in design time while absorbing the new technology offerings.

References

1. Bendas, J., "Design through Transformation," *Proceedings of the 20th Design Automation Conference*, 253–256. June, 1983.
2. Saunders, L.F., "The IBM VHDL Design System," *Proceedings of the 24th Design Automation Conference*, 484–490. June, 1987.
3. Dillinger, T.E., "A Logic Synthesis System for VHDL Design Descriptions," *Proceedings 1989 IEEE International Conference on Computer-Aided Design*, 66–69. November, 1989.
4. Elder, W.H., "An Interactive System for VLSI Chip Physical Design," *IBM Journal of Research and Development*, Volume 28, Number 5, 524–536. September, 1984.
5. Kirkpatrick, S., "Optimization by Simulated Annealing," *Science*, Volume 220, 671–680. 1983.

Input/Output Bus Using Fiber Optics

Describes the implementation of an attached input/output (I/O) bus using fiber-optic technology.

Bruce L. Beukema, Timothy R. Block,
Dennis L. Karst, Ronald L. Soderstrom, and
David W. Siljenberg

Introduction

Version 2 of the AS/400 system introduces the technology of fiber optics as an integral part of this product. The Version 2 system uses fiber-optic technology to attach input/output (I/O) buses on the 9406 System Units to increase I/O bandwidth and distance. This results in important customer benefits of increased I/O throughput, higher storage capacity, and longer distances between the processor and I/O devices.

A laser-based optical link card (OLC), operating at a 222Mbps (Mbps equals megabits per second) data rate, was designed at IBM Rochester to provide a cost effective, high data-rate link for the system I/O bus attachment [1,2]. Figure 92 on page 173 shows three of these optical link cards mounted on a local optical bus attachment card. Each OLC provides two completely independent optical links; therefore, the three OLCs provide six optical links for I/O bus attachment.

System I/O Bus Overview

Figure 91 on page 173 illustrates the structure of the 9406 I/O bus hardware. The system processor attaches to the I/O buses by means of a local bus attachment card, fiber-optic cables, and a remote bus attachment card. I/O controllers attach to the I/O bus and provide disk, tape, work station, communications, and local area network (LAN) I/O functions for the system.

The 9406 models provide the capability to attach up to seven I/O buses for increased throughput and capacity. The first I/O bus on the 9406 models resides within the same enclosure as the system processor. The remaining six I/O buses on 9406 models are attached with a fiber-optic connection to separate card enclosures.

9406 I/O Bus Hardware

Figure 92 on page 173 shows the details of the hardware content used in the connection of the attached I/O buses with fiber optics. The field-replaceable hardware components of the I/O bus subsystem include a local bus attachment card, fiber-optic cables, and a remote bus attachment card.

Local Bus Attachment Card

The local bus attachment card contains the logic used to attach the first I/O bus and optionally up to three bus interface units and three OLCs. The **bus interface unit** contains the logic used to buffer I/O data between main storage and the remote bus attachment card. The bus interface unit connects to an OLC. Each bus interface unit, together with an OLC, provides two I/O bus ports into the system processor. The OLC provides the interface to two dual fiber-optic cables and performs parallel-to-serial and digital-to-optical conversions.

Remote Bus Attachment Card

The remote bus attachment card contains two bus control units and one OLC. The **bus control unit** performs the direct attachment of one I/O bus to the OLC. Each bus control unit also attaches to the OLC to enable the transfer of I/O data between the remote bus attachment card and the local bus attachment card. The two bus control units are also interconnected on the card with printed circuit wiring to provide an alternative path between the bus interface unit and either of the bus control units. This alternative path provides redundancy in the fiber-optic I/O buses.

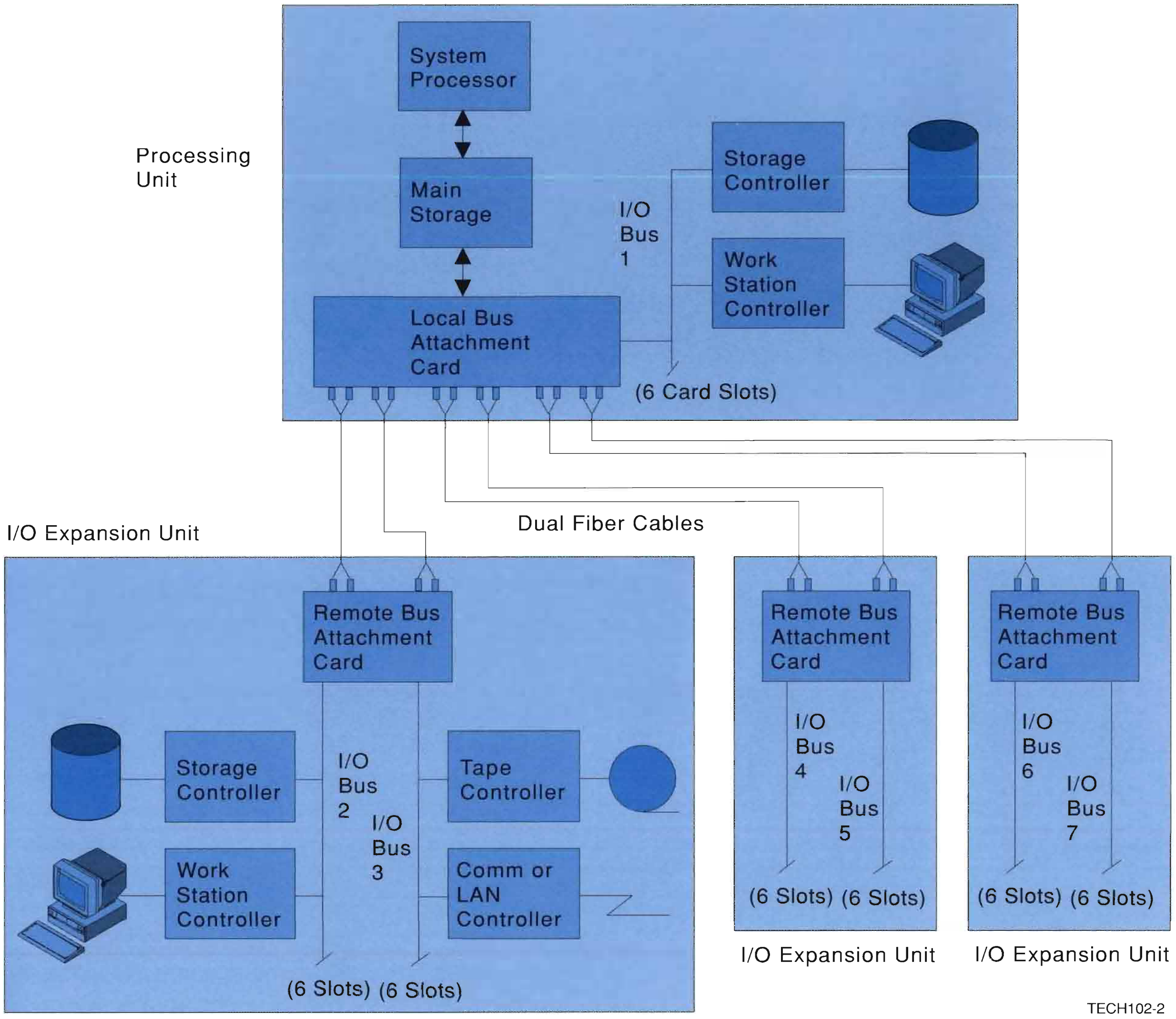


Figure 91. *9406 Model D80 with Seven I/O Buses*

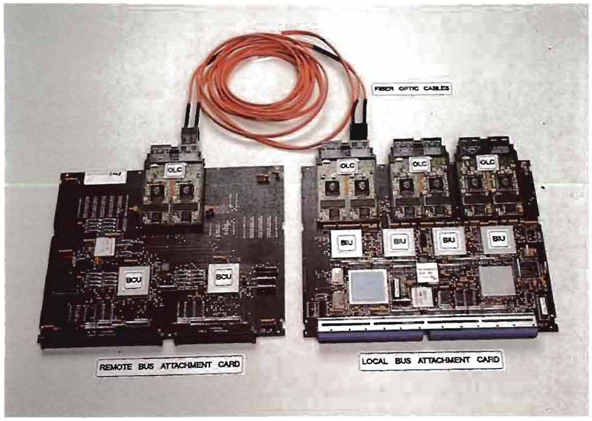


Figure 92. *Hardware Components in I/O Buses Attached Using Fiber Optics*

I/O Bus Enhancements

Numerous enhancements were made in the I/O bus design to provide customers with increased performance, more flexible configuration alternatives, and improved reliability.

Performance increases of the individual I/O buses combined with the additional bus attachment capability result in an increase of over 300% in I/O data throughput. Configuration alternatives are more flexible than previously possible with the use of fiber optics to connect additional I/O buses. Reliability improvements occur because fewer components are used to attach a greater number of I/O buses.

Fault Tolerance

Fault tolerance is built into the hardware that attaches the I/O buses. An alternative path between bus control units allows communications between a bus control unit and the bus interface unit on an alternative fiber-optic cable should a single failure occur in any OLC or fiber-optic cable component. Hardware mechanisms automatically retry data packets that incur errors. Software recovery procedures are called only when hardware retry procedures fail.

Optical Link Card Technologies

Key technology contributions leading to development of the OLC were made in the areas of packaging, optical components, optical connectors, optical fiber, electronics, and laser safety.

Package

The OLC package, including the optical duplex connectors, the cable, and the optical wrap plug feature, is shown in Figure 93.

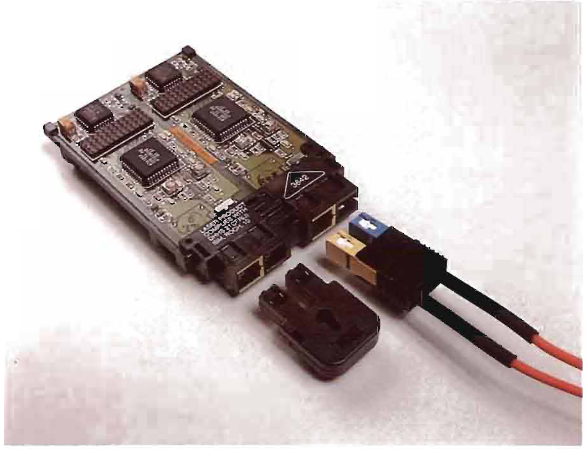


Figure 93. *OLC Package*

Surface-mount components on both sides of the circuit card allow the OLC to maintain a small form factor for high density packaging in comparison with industry equivalent function.

As shown in Figure 93, there are two identical, but electrically isolated, transmit and receive pairs on each card. Two 48-pin connectors are placed on the top side of the card such that the pins protrude through the card to the bottom side where they attach to the system card. This allows for a minimal 7-mm card-to-card spacing and a total card height of only 13 mm for low profile requirements. Four optical connectors are located at the edge of the card and extend into customer access areas when the OLC is placed on a system card.

Optical Components

For computer data links, the optical components (laser and detector) must be low cost and still provide high performance attributes, such as high data rate, low bit error rates, and reasonable transmission distances to include multiple building installations. The low cost requirement for components is the highest priority because a typical computer application has many transmitter and receiver ports. This is in contrast to telecommunications applications that employ few transmitter and receiver ports but many kilometers of optical fiber.

The semiconductor laser used for the OLC is a 780-nm device similar in structure to those devices that are currently in high volume production (that is, greater than 2 million per month) in the compact disc industry.

Semiconductor laser diodes offer many advantages over other types of transmitters in fiber-optic systems. One of these advantages is high data-rate capability. Another is the ability to launch high levels of optical power into the fiber. In addition, a laser transmitter requires low modulation current (on the order of 10 mA) due to high external efficiency.

The laser chip and a monitor photodetector, used to measure the laser output power, are packaged in a hermetically sealed can with a

5.6-mm diameter base. This small size allows for a compact laser receptacle. The laser receptacle is of a high coupling efficiency design (typically 50% into 50/125 µm fiber) using a single graded-index (GRIN) lens element and a precision fiber connector.

The receptacle housing and precision components are made of brass or stainless steel, which provide the precision and thermal stability required to maintain high optical coupling efficiency. The laser receptacle assembly is actively aligned during the assembly process to optimize the power coupled to the optical fiber with the laser package and GRIN lens being soldered or welded in place for good temperature stability.

The receiver photodetector is a 600-µm diameter silicon PIN photodetector. A **PIN** device consists of positive (P) and negative (N) regions separated by an intrinsic (I) region. Simple coupling techniques without the use of any additional lens result in a coupling efficiency of greater than 95%. Silicon PIN photodetectors are also low cost and, as an established technology, have a proven reliability record.

Example laser and detector receptacle packages are shown in Figure 94.

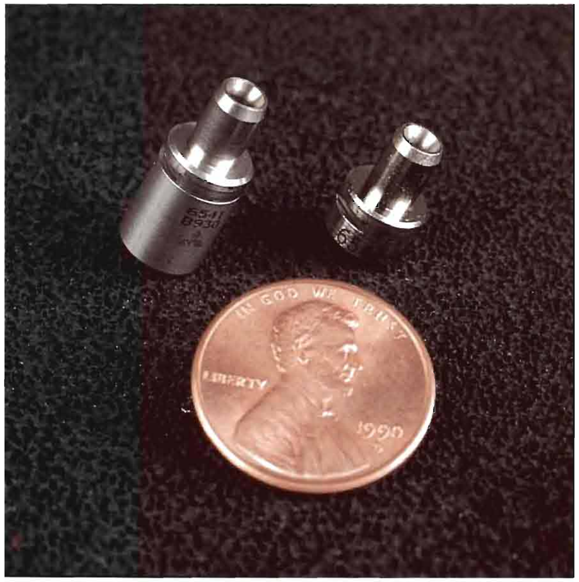


Figure 94. *Laser (Left) and Detector (Right) Receptacles*

Optical Connector

The optical connector of choice is a simple push-pull snap-in connector, commonly known in the industry as the SC type. This feature has many advantages over the standard twist-on connectors that are presently available. The simplicity of the connection is an important advantage for data communications applications because the connectors are accessible to the user. The SC connector snaps into place with an audible click that lets the installer know that the connector is properly in place. Similar to most of the twist-on connectors, the SC is keyed so that it can only be inserted the same way each time, ensuring consistent repeatability. The push-pull style of connector also increases the packaging density of the connector because there is no need to allow extra room for accessibility.

One of the most important features of the SC connector is the design of the **ferrule** (precision alignment tube). This design is referred to as a physical contact connection, which has become a standard in the connector industry. The SC ferrule is composed of a partially stabilized zirconia ceramic material. The end-face of the ferrule has a convex polish with an optimum radius of curvature of 20 mm. The use of this material and radius ensures good physical contact between two ferrules, which results in a low insertion loss and a high return loss by eliminating back reflections [3].

Optical Fiber

The preferred fiber is 50/125-µm multimode fiber because of its superior modal bandwidth compared to other types of multimode fibers and because of the large number of installations of this fiber, especially in Europe and Japan. An optional fiber is 62.5/125 µm multimode fiber, which provides similar performance at reduced distances. The fiber is cabled into a duplex construction using a plenum-rated jacket for flexibility at the installation location. The jacket material choice is critical to achieve the plenum rating while minimizing the allowable bend radius.

Electronics

The design approach of the electronics is to reduce cost using established technologies in plastic packages. The electrical schematic for a single port of a dual-port card is shown in Figure 95.

The parallel transmit data is received by a serializer module and serialized at 222Mbps. The serializer modulates the semiconductor laser with this data and controls the laser operating point. A phase-locked loop (PLL) generates a 222-MHz clock, needed for generating the serial data, from the 22.2-MHz transmit clock. There are also safety circuits in the laser drive function to turn the laser off if an on-card fault occurs, which could produce an unsafe power level.

The incoming light is received by the photodiode and the signal is amplified by the transimpedance preamplifier. The deserializer module uses a PLL to generate a 222-MHz clock from the serial data. The deserializer then clocks the data out in 10-bit words at 22.2 MHz. The shift register has a byte synchronization detector that recognizes a unique character so that the complete bytes can be unloaded from the shift register without being fragmented. The transition detector and the dc detector detect a minimum ac and dc level of light entering the photodiode. This is used by the open fiber control (OFC) module to detect when the fiber path is open.

The transimpedance preamplifier is implemented in a 6-GHz bipolar process. It is packaged in a 14-pin plastic small outline integrated circuit (SOIC) package. The serializer and deserializer are implemented in a 4-GHz trench-isolated bipolar process. They have 800 and 1,200 transistors, respectively, and are packaged in 44-pin plastic-leaded chip carriers (PLCC).

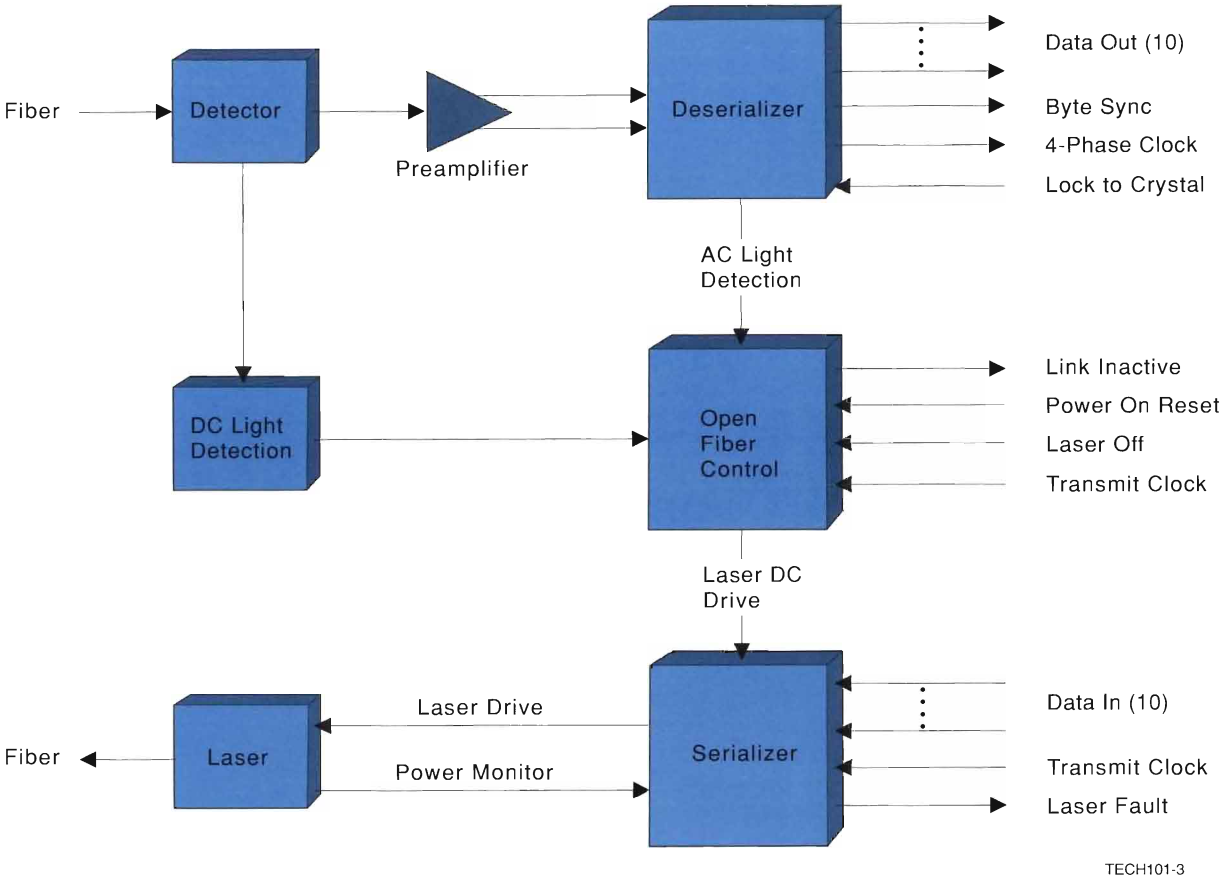


Figure 95. *Electrical Schematic Block Diagram*

The OFC is a safety interlock to shut down the link if both fiber paths are not closed. The OFC module pulses the laser at a low duty cycle while a fiber is open. This produces Class 1 optical power in the open port. When the fibers are reconnected, the OFC returns the laser to continuous power. The OFC is implemented in 2,200 complementary metal-oxide semiconductor (CMOS) circuits. It is packaged in a 28-pin PLCC.

The bus interface unit is implemented in an 0.8-µm semicustom CMOS chip using 40,000 circuits. It is packaged in a ceramic-pin-grid array.

Laser Safety

When designing an optical link with a laser transmitter, laser safety standards and legal regulations need to be understood and incorporated into the development process. This ensures a final laser product that is both safe and in conformance with legal regulations throughout the world.

The OLC uses the OFC function in conjunction with redundant circuitry and laser safety features incorporated in the laser drive circuits. The OLC laser product has been certified to be in conformance with worldwide laser safety regulations as a Class 1 laser product. Class 1 is the least restrictive classification for laser products and requires minimal labeling and documentation.

Conclusion

The use of fiber-optic technology has enabled the development of 9406 models with enhanced I/O attachment throughput and flexibility. I/O attachment devices may now be located remotely from the processing unit with performance nearly equal to those attached locally. An affordable cost for the feature is achieved through the use of low-cost components and packaging, namely, short-wavelength laser-based sources and surface-mount assembly. Overall system reliability is improved through fault tolerance and through a reduction in the number of components.

References

1. Soderstrom, R.L., Block, T.R., Karst, D.L., and LU, T., "An Optical Data Link Using a CD Laser," *Proceedings of SPIE: High Speed Fiber Networks and Channels,* to be published in Winter 1991, Boston, MA. September, 1991.
2. Soderstrom, R.L., Block, T.R., Karst, D.L., and Lu, T., "CD Laser as a Fiber Optic Source for Computer Data Links," *Proceedings of SPIE: Fiber Optic Datacom and Computer Networks,* Volume 991, Boston, MA, 179–182. September, 1988.
3. Sugita, E., Iwasa, K., and Shintaku, T., "Design for Push-Pull Coupling Single Mode Connectors Featuring Zirconia Ceramic Ferrules," *ECOC '86,* 141. 1986.

Designing Twinaxial Work Station Controllers for Performance

Discusses the performance-oriented design of the AS/400 twinaxial work station controller.

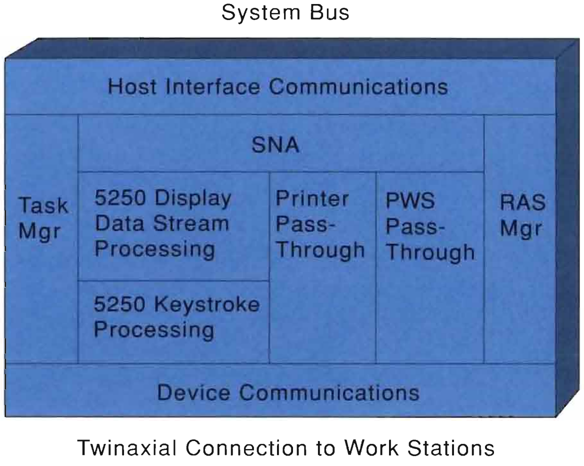
Harvey G. Kiel and Ricky M. Peterson

Introduction

Local work station controllers (WSCs) handle the exchange of information between host applications and local work stations. Data stream commands and data flow to and from the host, and input/output (I/O) commands and data flow to and from the user's work station (typically a nonprogrammable work station (NWS), personal computer, or printer).

WSC design is especially important to a user; it handles the information with which the user interacts. Because performance problems, such as sluggish response time for a keystroke or inconsistent printer speed, are causes for customer dissatisfaction, special consideration should be given to the WSC and its related interfaces.

The twinaxial work station subsystem (specifically, the WSC) consists of several processes, such as host interface communications, 5250 display data stream processing, and device communications. Figure 96 shows the major WSC software components. Throughout the WSC design, performance implications are given a high priority. This article discusses specific performance enhancements relative to each of the components.


System Bus
Host Interface Communications
SNA
Task Mgr
5250 Display Data Stream Processing
Printer Pass-Through
PWS Pass-Through
RAS Mgr
5250 Keystroke Processing
Device Communications
Twinaxial Connection to Work Stations

PWS = Programmable Work Station
RAS = Reliability, Availability, Serviceability
SNA = Systems Network Architecture

TECH071-2

Figure 96. *Twinaxial Work Station Controller Structure*

Host Interface Communications

For the WSC to offer functions external to the system processor, the WSC must receive commands and data from the host, and information must flow back to the host from the WSC. The content of the data flowing to or from the WSC depends on the type of work station attached. For an attached NWS, the host sends a 5250 display data stream that describes the type of screen to present to the user. After the user enters data at the keyboard, information that describes the user's input is sent back to the host. For printers, the host sends a printer data stream that is required for the printer to produce the necessary print image, and little information flows back to the host. Personal computers can function as emulated NWSs or as programmable work stations (PWSs). The former acts the same as an NWS. The latter can act as the WSC for that personal computer or as a separate computer system, thus receiving data from the host (passed through the WSC). PWSs that act as a separate computer system may request the WSC to transfer data between the AS/400 system and the PWS. The rate that data flows to or from the WSC also depends on the work station being used. NWSs may receive data infrequently. Printers and PWSs may require data to flow for long periods of time.

AS/400 WSCs communicate with the host using large frame sizes: 4,608 bytes for NWSs and PWSs and 2,048 bytes for printers. This design significantly reduces host, WSC processor, and system bus utilization during host and WSC communications. AS/400 WSCs are able to support large frame sizes through the use of dynamic memory allocation. NWSs require an entry-field definition table (2,048 bytes) but require communications buffers only for brief periods of time and usually only one buffer at a time. PWSs and printers do not require an entry-field definition table, but they may require several communications buffers for a longer period of time.

To minimize overhead during WSC-to-host communications, WSC components (for example, 5250 data stream processing and PWS pass-through) reserve space at the beginning of each data buffer for bus specific data. The host interface communications can then add its data at the beginning of the buffer. As a result, data is never copied within the buffer or between buffers and yet two dissimilar types of information are sent to the host in a single direct memory access (DMA), reducing WSC processor and system bus utilization.

Task Manager

The task manager component, or control program, handles the distribution of tasks between other components, work queues, and storage allocation.

The task manager component calculates processor utilization each six seconds by using an idle loop counter and the following formulas:

maximum idle count = span of test / processing time of idle loop

processor utilization = (maximum idle count–actual idle count) / maximum idle count

Processor utilization is characterized by six ranges: 0%–10%, 10%–20%, 20%–30%, 30%–40%, 40%–75%, or 75%–100%. A maximum adjustment of one utilization range is allowed per utilization calculation to smooth an uneven work load. The resulting utilization range is made available to all components in the WSC, which allows tuning based on work load. Optional task trace and automatic adjustment of polling rates to work stations are two examples of tuning based on workload. Both examples are described in this article.

AS/400 WSCs support **task trace**, which aids in debugging problems. Initially, AS/400 WSCs start task trace only when the system sends a Start Trace command. The WSC licensed internal code may detect a WSC problem, in which case, the WSC automatically causes an unsolicited dump. The customer is requested to send the dump to IBM for analysis. Unless the customer specifically turns on WSC trace, task trace is not running. A dump with task trace running usually provides enough information to debug the problem; however, a dump without task trace usually does not. The WSC could have been designed such that task trace was always on, but task trace typically increases processor utilization between 5% and 15%. The solution is to use the calculated processor utilization range to determine if optional task trace should be running. If the WSC processor utilization range is 20% to 30% or less and task trace is not running, the WSC starts optional task trace. Performance degradation of task trace is minimal under this type of WSC work load. If the WSC processor utilization range is 40% to 75% or higher and optional task trace is running, the WSC stops optional task trace. If the system sends a Start Trace command, task trace runs independently of WSC processor utilization.

5250 keystroke processing tasks have highest priority and preempt processing of other tasks, such as host communications and 5250 data stream processing. With this approach, keyboard overruns become extremely rare and keystrokes that require only WSC processing appear to be processed instantaneously. **Keyboard overruns** occur when the WSC is unable to process keystrokes as fast as the user is entering them. When a keyboard overrun occurs, an operator error is shown and the user must press the Reset key.

Optimizing the performance of critical code paths is not new, but it significantly improves overall WSC performance. A logic analyzer is used to analyze path lengths of typical operations, and the licensed internal code is tuned where appropriate. The most critical areas are parts of the task manager, data keystroke processing, the interface between the system manager and keystroke processing, and data character processing for both inbound and outbound 5250 data streams.

SNA

The AS/400 WSC Systems Network Architecture (SNA) function supports logical unit (LU) type 4 for printers and LU type 7 for NWSs and PWSs. The SNA function establishes and maintains sessions between application programs and work stations. The SNA function also provides a mechanism for transporting data across these sessions.

Sessions for SNA LU type 4 or 7 are half-duplex protocols. The half-duplex protocol is not a problem for NWSs or printers because the

SNA change direction indicator (CDI) can be managed properly by the host and WSC SNA components. In the PWS case, the LU 7 CDI is difficult to manage because of multiple-PWS-session capability. This results in many SNA SIGNAL commands to request the CDI and null frames to pass the CDI. Originally, the AS/400 system used the standard half-duplex LU 7 to support PWSs. Problems were identified including delays on send operations in both the host and WSC SNA components (when the CDI was at the wrong side) and extra host and WSC processor work load as a result of CDI management. To address these problems, the WSC designers developed a duplex version of SNA session type 7 specifically for PWS attachment (this design does not function for NWSs or printers). The new design allows the host and WSC SNA components to send frames as necessary. It uses no CDI, no SIGNAL commands to request CDI, and no null frames to pass CDI. The new design significantly improves performance.

Reliability, Availability, and Serviceability Manager

The reliability, availability, and serviceability (RAS) manager handles logging of errors, monitoring work station detection, and collecting and reporting performance data.

AS/400 WSCs report temporary twinaxial protocol errors to the host for logging. Logging all temporary twinaxial protocol errors on a system with a substandard twinaxial cable or substandard work stations seriously degrades WSC and host performance. The WSC allows a maximum of five temporary errors per work station per 30-minute interval. This results in enough information about the problem without further degrading performance with excessive error logging.

AS/400 local WSCs support automatic configuration. The WSC polls all twinaxial addresses and informs the host of any work station powered on or powered off. Unsatisfactory performance to users of powered-on work stations is minimized by polling a single, unused twinaxial address at a very slow polling rate. Also, WSC changes unused twinaxial addresses at a rate determined by the number of powered-on work stations, resulting in fewer changes between addresses for a WSC with many work stations attached.

AS/400 WSCs generally have good performance characteristics; however, there may be some cases of an overloaded WSC. Also, customers studying system performance may want to analyze WSC performance when attempting to predict future system performance. The AS/400 system provides performance tools that allow customers to analyze and improve system performance. AS/400 WSCs have a built-in performance monitor that captures and reports performance data to the host. Each WSC reports estimated processor and twinaxial utilizations, and individual system response times for each NWS.

5250 Display Data Stream Processing

NWS response time for keystrokes requiring host processing is the sum of WSC GET time, host time, and WSC PUT time. **GET time** is the time required to process a keystroke requiring host processing, read the NWS screen, and send an inbound 5250 display data stream with processed entry-field data to the host processor. **PUT time** is the time required to process an outbound 5250 display data stream, update the entry-field definition table, update the NWS screen, and unlock the keyboard. The data stream processing time for NWSs can be considerable. Information in the outbound data stream must be converted to NWS commands in a format that the NWS can understand. When the NWS sends information requiring host processing to the WSC, that information must be converted to an inbound data stream that the AS/400 display handling programs can understand.

WSC GET and PUT processing can be a significant portion of overall system response time. David Andrews of ADM, Inc. stated the following: "One of the important design features of the AS/400 is a newly designed work station controller significantly faster and more sophisticated than those offered with the S/36 or S/38. This new controller is a major contributor to the faster response times...."[1].

There are basically two types of PUT operations: full screen, which creates a new screen format, and partial screen, which updates an existing NWS screen. Average PUT operations contain between 50 and 100 data stream elements, with some containing many more elements. Prior to AS/400 systems, WSCs processed each element individually, whereas most elements map into one or two NWS commands. NWS commands usually require between 2 and 5 ms of NWS service time. In some cases, AS/400 WSCs create temporary screen buffers in WSC storage, process PUT elements into the screen buffer, and finally update the NWS screen with only two to four NWS commands.

The WSC has three methods of processing PUT operations:

- Full-screen PUT processing
- Large partial-screen PUT processing
- Small partial-screen PUT processing

Full-screen PUT processing is done for a data stream that begins with a Clear Unit 5250 data stream command. The WSC creates a temporary screen buffer and clears the buffer as part of Clear Unit processing. All subsequent data stream commands and orders are processed into the screen buffer. When the end of the PUT data stream is found, the WSC rewrites the entire NWS screen. This method of PUT processing is probably the most visible WSC performance improvement over other WSCs. Prior to AS/400 systems, WSCs required several hundred milliseconds to process typical full-screen PUT operations whereas NWS commands are sent throughout PUT processing. WSCs now process most full-screen PUT operations in less than 100 ms. The screen is updated at the end of PUT processing (appearing to be an instantaneous screen update). Large frame sizes between the host and WSC are beneficial because most PUT operations fit into a single frame.

Large partial-screen PUT processing is done for a large data stream that does not contain a Clear Unit data stream command. The WSC reads the entire NWS screen into a temporary screen buffer and processes all subsequent data stream commands and orders into the screen buffer. The NWS screen is written from the first byte changed to the last byte changed when the end of the PUT data stream is found. This method of PUT processing provides significantly better performance than processing each data stream element individually.

Small partial-screen PUT processing is done for a small data stream that does not contain a Clear Unit data stream command. The WSC processes each data stream element individually by sending NWS commands. Prior to AS/400 systems, WSCs essentially operated in this mode all the time. This method of PUT processing offers better performance than large partial-screen PUT processing for small data streams because the number of NWS commands is limited. Because reading the entire NWS screen requires about 40 ms, data streams with less than 10 to 15 typical NWS commands can be processed faster than with the large partial-screen PUT processing method.

When a user presses a keystroke requiring host processing, the WSC performs GET processing, which includes reading data from NWS screen entry fields. Prior to AS/400 systems, WSCs always read each individual entry field separately, requiring two NWS commands. The time required for the NWS commands becomes significant when an NWS screen contains many entry fields (up to 256 entry fields are allowed). AS/400 WSC licensed internal code uses the trade-off between multiple read operations and reading extra data between entry fields to optimize performance.

There are two types of read operations, unformatted (the NWS returns screen data) and formatted (the NWS performs processing on the screen data before returning the read data to the WSC). The NWS time for unformatted read operations is small. For an unformatted read operation, the WSC reads the NWS screen into a temporary screen buffer from the first entry field to the end of the last entry field. This results in fewer NWS commands and improved performance in nearly all cases. The NWS time for formatted read operations is related to the size of the read data, which can be significant. The WSC may group read operations together when a formatted read operation is going to be used (word processing mode in certain cases). The WSC groups two or more read operations into one when the extra NWS and twinaxial transmission times are less than the overhead of an extra read operation.

5250 Keystroke Processing

NWS users interact with AS/400 applications with keystrokes. The keystroke entered may or may not show up on the NWS screen and may or may not be sent to the host. Keystroke processing is performance-sensitive. For keystrokes processed completely by the WSC, users expect the keystroke to be processed immediately. A visible delay is unacceptable. If the keystroke response time appears to be sluggish, customer dissatisfaction is imminent.

The WSC has added support for type-ahead. **Type-ahead** involves buffering keystrokes in the WSC when the keyboard is locked, and processing them when the keyboard unlocks. Type-ahead can significantly improve an expert user's productivity because "think time" can be significantly reduced on displays well known to the user. **"Think time"** is from the time the keyboard unlocks after an NWS screen has been written (keystrokes can be processed) until the user has completed entering keystrokes. Type-ahead is useful when a user is working with documents in word processing mode (WP mode). If a document is too wide for the NWS screen, type-ahead can be used to make it easier for the user to see the parts of the document to the right or left of the NWS screen. The WSC sends a request to the system for more data when the cursor reaches the left or right edge of the NWS screen and more data exists beyond it. As a result, the user is not interrupted while power typing wide documents.

Printer Pass-Through

In general, the WSC simply passes printer data streams through to the specified printer.

Two critical areas of printer performance are printer throughput (keeping printers printing at rated speed) and minimal degradation to other users on the WSC caused by printers. Performance testing results show that AS/400 WSCs can support a total aggregate printer throughput (one or more printers) of about 3,500 lines per minute (lpm) with minimal degradation to other users on the WSC.

AS/400 WSCs support two modes, normal WSC mode and dedicated printer WSC mode. Each time a work station is powered on or off, the WSC checks all attached work stations that are powered on. If only printers are attached, the WSC is considered a dedicated printer WSC. Printers are then polled at fast rates allowing a maximum single-printer throughput much higher than 3,500 lpm. This solution prevents printers from degrading performance of other users on the WSC and allows high printer throughput on a dedicated printer WSC.

PWS Pass-Through

PWSs (running PC Support/400) have several critical areas of performance including throughput for file transfer operations, response time for keystrokes requiring host processing when running AS/400 applications, and minimal degradation caused by PWSs to other users on the WSC. Prior to AS/400 systems, personal computers were attached to the system using existing 5250 interfaces. Each 5250 emulation and file transfer session uses a dedicated twinaxial address and appears as a separate 5250 NWS to the WSC. There are several performance inefficiencies with this design. For example, because file transfer operations fit within the context of an NWS screen (limited to a maximum transfer size of 1,904 bytes before an end-to-end handshake is required) and pseudokeystrokes are used to trigger a read or write operation to the pseudo-NWS screen, file transfer throughput is limited. Also, a single personal computer can use several twinaxial addresses, increasing both WSC twinaxial and processor utilization due to polling each of these addresses.

A new 5250 data stream protocol between the host and WSC was designed for PWS users. The protocol essentially turns the WSC into a pipe, simply passing data between the host and PWS. The WSC is transparent to the type of data, and handshaking between the host and the WSC is kept to a minimum.

Prior to AS/400 systems, PWSs would commonly use more than one twinaxial address to have more sessions active. If more than one twinaxial address is used, the number of devices attached to the WSC is reduced. To make the connection between the WSC and the PWS more efficient, AS/400 WSCs require only a single twinaxial address independent of the number of sessions running on the PWS.

To minimize handshaking between the WSC and the PWS, a new protocol between the WSC and PWS is used. The first two bytes of download data contain the length of the download data. The PWS requests that data be sent to the host by passing a scan code to the WSC on a poll. A scan code indicates a keystroke for an NWS. A scan code from the PWS approximates the length of the data to be sent from the PWS to the host (a scan code of 1 is equivalent to 2 through 64 bytes; a scan code of 2 is equivalent to 65 through 128 bytes; a scan code of 3 is equivalent to 129 through 192 bytes; and so on). The WSC then reads the data from the PWS. If the data length is less than 1,024 bytes, the WSC reads the data in a single read operation. The first 2 bytes of data contain the exact length of data to be sent to the host. The new design also emphasizes overlapped processing. WSC processor and twinaxial utilizations are also reduced by the new design.

Device Communications

The actual communications activity between the WSC and work stations takes place over a 1Mbps (Mbps equals megabits per second) twinaxial communications line. While this may seem fast, there may be a considerable delay if a lot of data flows for a given transaction or if there is significant queuing on a single set of hardware twinaxial drivers and receivers. Also, twinaxial protocol requires 16 bits per byte of data sent or received. This overhead reduces the maximum throughput to 62.5KBps (KBps equals kilobytes per second). Hardware logic in the WSC and the WSC licensed internal code are optimized for performance.

AS/400 WSC twinaxial hardware supports the concept of an **automatic-poll chain**. Prior to AS/400 systems, WSCs issued a request to the twinaxial hardware adapter for each poll. The WSC microprocessor was then interrupted when a poll response was received. The AS/400 WSC automatic-poll chain controls polling for keystrokes from NWSs for new status information from PWSs or printers and for command completion for a previously sent command. Each request on the chain has its own polling rate (using a skip count), allowing the WSC to set individual work station polling rates. AS/400 twinaxial hardware interrupts the WSC microprocessor only when a keystroke or new status is received, a previously sent command has been completed, or an error has been detected. There are two significant advantages to hardware support for an automatic-poll chain: WSC processor utilization is significantly reduced and the twinaxial interface to work stations is used more efficiently. These occur because the hardware is able to proceed from request to request on the automatic-poll chain faster than if the licensed internal code issued each individual poll request. Automatic-poll support in WSC hardware allows the WSC processor to handle requests for work stations while the WSC hardware concurrently polls work stations.

Polling rates directly affect performance. Twinaxial communications generally have a higher capacity than the WSC processor. The WSC adjusts polling rates to trade off twinaxial latency times against twinaxial and processor utilizations. **Twinaxial latency** is the time from when a work station is ready to inform the WSC of an event (or change in status) until the WSC actually polls the work station and detects the event. Fast polling rates optimize performance by minimizing twinaxial latency times on a lightly loaded WSC. However, faster polling rates increase twinaxial utilization as well as processor utilization, degrading performance on a heavily loaded WSC. Processor utilization is affected because hardware access to automatic-poll requests kept in WSC storage briefly prevent the WSC processor from accessing storage. Therefore, polling rates are adjusted using a closed feedback loop based on the WSC processor utilization range described previously. The result is a WSC that has optimal performance across light, medium, and heavy work loads because processor and twinaxial utilization is balanced.

WSC developers measured WSC performance characteristics at various polling rates. A performance model was then tuned, and response time versus work load was plotted. Figure 97 on page 184 shows a sample of the response time curves for a heavy WSC work load. The curves show three polling rates: 1 ms with a skip count of 10, 3 ms with a skip count of 6, and 6 ms with a skip count of 4. As expected, fast polling optimizes performance on a lightly loaded WSC but degrades performance on a heavily loaded WSC. The goal is optimized performance across varying work loads; therefore, the lowest response times and cross-over points are of the most interest. The cross-over points are marked with arrows and show where polling rates would be adjusted. Although only three are shown in Figure 97 on page 184, the WSC actually uses six different polling rates based on the processor utilization range (see Table 3 on page 184).

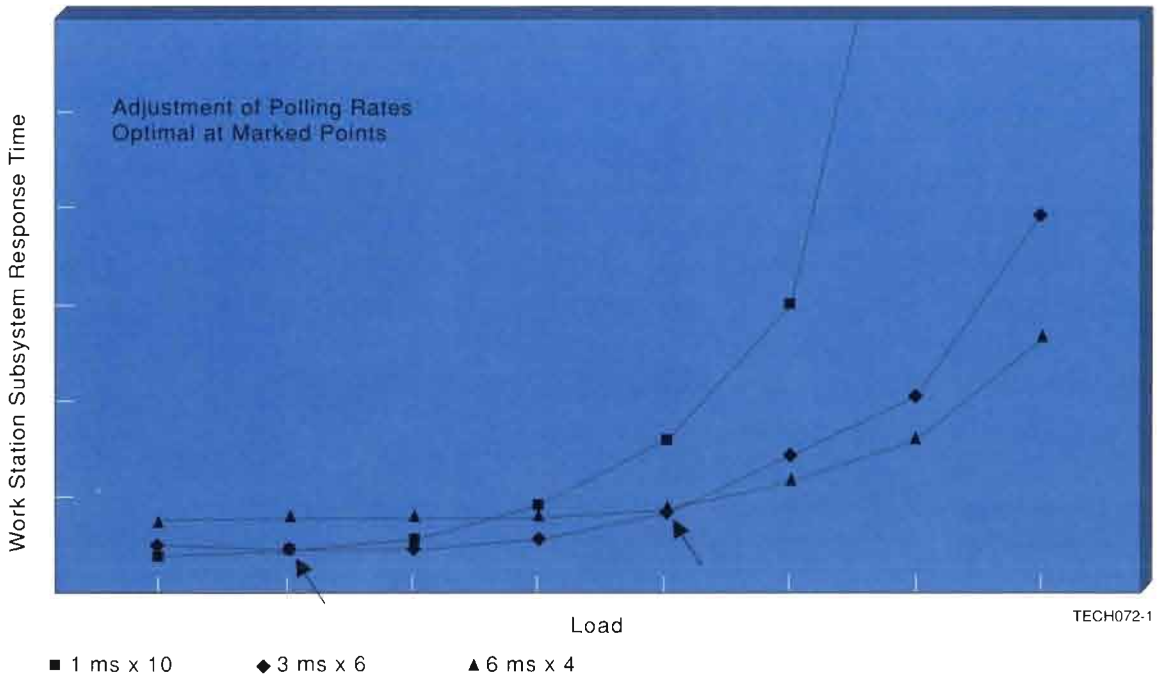


Figure 97. *Adjustment of Polling Rates Based on Processor Utilization*

AS/400 WSCs gradually reduce polling to inactive work stations. Studies of AS/400 system use show that about 40% of powered-on NWSs are actually inactive for significant periods of time. The percentage for inactive PWSs is probably even higher because PWS users can run PC applications, making the PWS appear inactive to the WSC. Because WSC hardware supports an automatic-poll chain where each request on the chain has its own polling rate, the WSC can gradually reduce polling to inactive NWSs and PWSs. This results in lower WSC twinaxial and processor utilization and improved performance for active users. Polling rates for keystroke and new status polling can be defined as active or inactive. Specifically, every 12 seconds, the polling rate to all NWSs and PWSs is increased slightly unless they are already at the inactive polling rate. The NWS or PWS polling rate is set back to the active rate any time a keystroke is received, a new status from a PWS is received, or any data is received from the host for the NWS or PWS. After a keystroke is detected and processed, the next poll to that NWS or PWS is issued almost immediately instead of delaying the polling rate. This allows the WSC to catch up if it fell behind in keystroke processing. The inactive polling rate is reached after about one minute of inactivity from an NWS or a PWS (five adjustments). Table 4 on page 185 shows active and inactive polling rates based on processor utilization.

Table 3. Adjustment of Polling Rates Based on Processor Utilization

Processor Utilization Range	PWS Polling for Keystroke or New Status (ms)[10]	Printer Polling for New Status (ms)	Command Completion Polling (ms)
0%–10%	10	12	1.0
10%–20%	12	18	1.5
20%–30%	14	24	2.0
30%–40%	18	30	3.0
40%–75%	20	40	4.0
75%–100%	24	54	6.0

[10] Polling rates to NWSs and PWSs are partially based on the last activity from the NWS or PWS (described in the following paragraph). The rate shown in Table 3 reflects an active NWS or PWS.

Table 4. Gradual Reduction of Polling of Inactive Work Stations

Processor Utilization Range	Active Polling Rate (ms)	Inactive Polling Rate (ms)
0%–10%	10	15
10%–20%	12	18
20%–30%	14	20
30%–40%	18	27
40%–75%	20	36
75%–100%	24	48

Twinaxial printers have always had a 256-byte interface with WSCs. WSCs send printer data to the printer in a 256-byte block, wait for an acknowledgment that the data has been received, wait for an indication that the printer is ready to accept another 256-byte block, and send the next block. AS/400 WSCs significantly reduce the handshaking between the WSC and the printer by using a larger block size for selected printers. Some newer twinaxial printers support block sizes of 1,024 bytes and 256 bytes. AS/400 WSCs take advantage of the larger byte blocks, resulting in reduced processor and twinaxial utilization.

Conclusion

The twinaxial work station controller was designed as a high performance I/O processor. Improvements over previous work station controllers include more attached work stations, faster response time of keystrokes and NWS transactions, and higher throughput of printer and PWS transactions.

AS/400 developers not only designed the work station subsystem with performance in mind, but also made the hardware and software more adaptive to the environment. This allows consistent response time and throughput for a wide variety of configurations and work loads.

Reference

1. Andrews, D., Martin, J., Elms, T., Simeone, D., et al, *The AS/400 Revolution*. May, 1990.

Thin-Film Magnetic Disks

Describes the thin-film disk technology used in IBM disk units.

Kenneth E. Johnson

Introduction

IBM thin-film magnetic disks represent a major advance in the design and production of rigid-disk recording media. In the quest for higher recording densities on rigid disks, thin-film media present a major advantage over the existing particulate disk technology [1]. Uniform films can be easily deposited with thicknesses of only several hundred angstroms (Å), hence the origin of the term **thin film.** Thicknesses in this range are essential for high density magnetic recording.

Thin-film disks appeared in the marketplace in the early 1980s, and their magnetic benefits were immediately obvious. However, because of the design challenges surrounding disk durability, some reluctance emerged regarding incorporating thin-film media into disk unit designs. Implementing thin-film media into IBM products required significant advances in thin-film disk tribology and thin-film disk corrosion resistance. Thin-film disk wear mechanisms are poorly understood and, combined with the susceptibility to corrosive degradation, disk life can be sharply curtailed. The presence of a high fraction of chromium (Cr) in the sputtered magnetic film produces acceptable corrosion resistance. The addition of a thin sputtered carbon overcoat on top of the magnetic film, followed by the application of approximately one monolayer of lubricant, extends thin-film disk media life. The thin-film disks used in the IBM 0671, 0661, and 0681 disk units are the first such disks from IBM development laboratories to fulfill the stringent requirements for inclusion in an IBM disk unit.

Enhancements to the thin-film disk structure, its manufacturing process, its magnetic performance and mechanical durability, and its corrosion resistance have been made. (For more technical detail on thin-film magnetic disks, see the excellent review entitled "Thin-Film Recording Media" written by T.C. Arnoldussen [2].)

Disk Structure and Manufacturing Process

IBM thin-film magnetic disks consist of several layers of metallic and nonmetallic films. Well-known techniques are used to achieve the desired depositions. First, the aluminum substrate is chemically plated with a relatively thick layer of nickel that is brought to a desired finish using abrasive finishing processes. The magnetic layer and the associated underlayer and overcoat are then applied by a vacuum process known as **magnetron sputtering**. Chemical plating can also be used for magnetic film deposition, but sputtering was chosen because of the ease in sputtering ternary magnetic alloys of different compositions. This versatility in choosing magnetic alloys is needed for the thin-film disk used in current and future IBM products.

As a general example of a thin-film disk structure, Figure 98 on page 187 shows a cross-section view of an IBM 0671 thin-film disk. As in the majority of rigid-disk products, the disk's magnetic and overcoat structure is placed on an aluminum (Al) alloy substrate. IBM has shipped thin-film disks in three sizes: the 0671 130-mm disk with an inside diameter of 40 mm and a thickness of 1.9 mm, the 0681 130-mm disk with an inside diameter of 40 mm and a thickness of 1.3 mm, and the 0661 disk with an inside diameter of 25 mm and a thickness of 0.8 mm. The aluminum is layered with a 14-µ deposit of hard nickel-phosphorous (NiP) using a process known as electroless plating, which is a chemical plating process that does not require external batteries or power supplies. This surface is equivalent in hardness to carbon steel (600 kg/mm^2) and serves as a firm base for the thin magnetic layer. In addition, the NiP surface allows for a smooth and controlled surface finish after abrasive processing, which a soft aluminum surface alone could never provide. The surface morphology of the polished and textured NiP is replicated in the next three layers of the disk structure. The top surface must be free of asperities to allow recording heads to fly closely and to min-

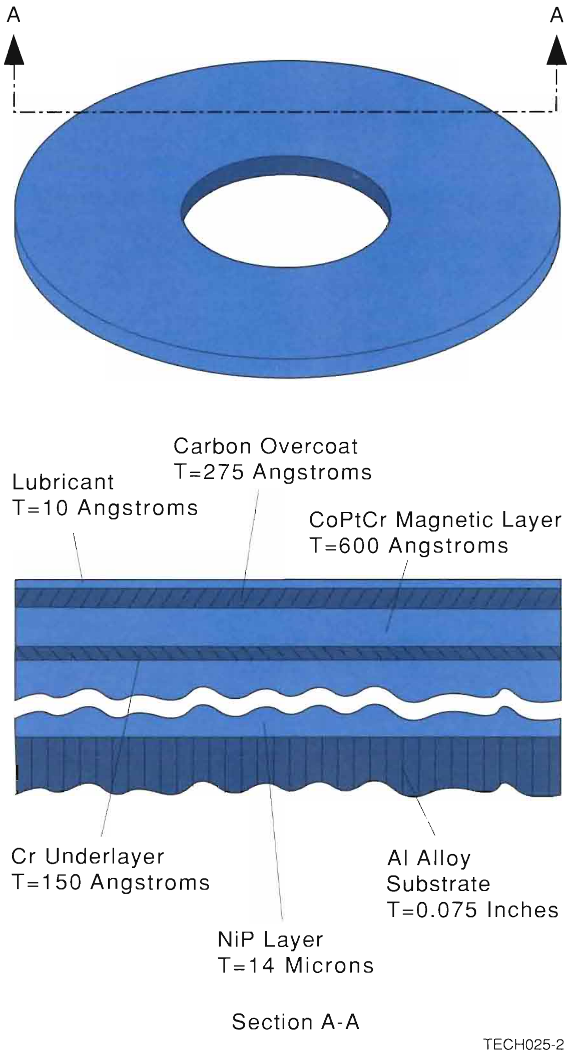


Figure 98. *Cross-Section View of an IBM 0671 Thin-Film Disk*

imize head-to-disk interactions that can accelerate disk wear.

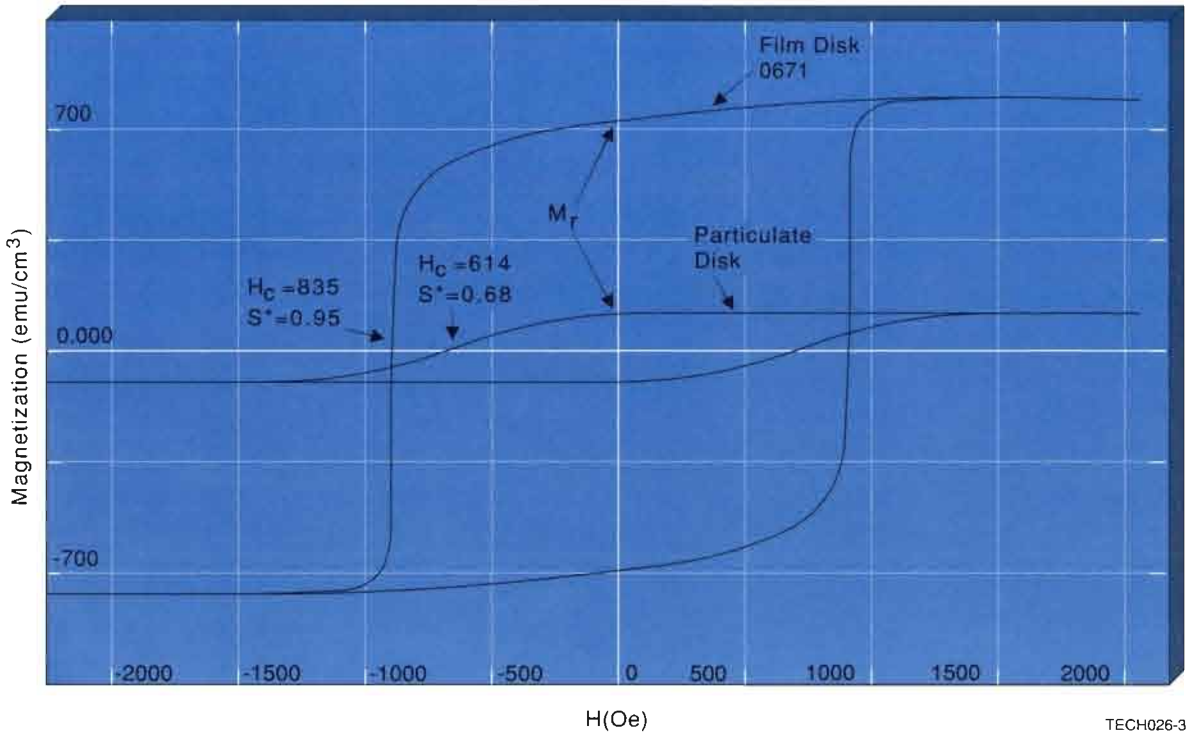
Next, the thin-film NiP surface is polished to a high degree of smoothness (25 Å) and then given a circumferential texture (75 Å measured radially). This texture is readily identified by the eye and serves to reduce stiction (the static friction force resisting initial disk motion) and friction, while at the same time improving magnetic properties because of preferential magnetic-grain alignment along the circumferential grooves. Although a surface texture is purposely added, the resultant disk surface still allows a recording head to fly at 175 nm at its inner diameter without encountering any interference or interactions.

The essence of the IBM thin-film disk is built into the next four layers. After the hard NiP surface is polished and textured, a Cr underlayer followed by the cobalt-platinum-chromium (CoPtCr) magnetic layer is sputter-deposited in a vacuum chamber. A sputtered carbon (C) overcoat is subsequently deposited in a second vacuum chamber of the same sputtering apparatus. This entire process is accomplished using a moving vertical pallet containing twenty-three 130-mm disks or forty-five 95-mm disks, which are sputtered over the course of several minutes. This process can produce a large quantity of disks at low cost with uniform film thicknesses and magnetic properties. After emerging from the sputtering step, a lubricant layer is deposited by dipping the disks into dilute solutions containing hydrocarbon or fluorocarbon.

Disk Magnetics

Producing a high quality magnetic disk involves more than sputtering a uniformly thin metallic layer. Magnetic anisotropy (the tendency for a material to magnetize in a specific direction) must be rigidly controlled to prevent degrading effects, such as modulation in the read-back signal [3]. One of the keys to producing a suitable magnetic film for longitudinal recording (the recording of bits parallel to the direction of recording) is to ensure that the magnetic anisotropy is in the plane of the disk. This preferential horizontal alignment is necessary to produce high squareness hysteresis loops that, in turn, lead to high-density recording capabilities.

This alignment can be controlled by several factors, but a major contributor is the identity, orientation, and thickness of the underlayer as described by J.K. Howard and others [4,5]. Depositing the thin (100 to 300 Å) Cr underlayer at a high rate (2,000 Å per minute) results in a crystallographic-preferred (110) orientation that allows the CoPtCr film to grow epitaxially, with its easy axis of magnetization in plane. Figure 99 on page 188 shows a magnetization (M) versus magnetic field (H) hysteresis loop of a CoPtCr thin-film disk compared to a particulate disk. The thin-film disk's curve exhibits a high degree of squareness compared to the particulate film. This is a result of the preferred growth characteristics influenced by the underlayer and the general properties of magnetic-metallic thin films.



TECH026-3

Figure 99. *M-H Hysteresis Curve for an IBM 0671 Thin-Film Disk and a Particulate Disk*

Two squareness parameters, the coercive squareness (S*) and the remanent squareness (S), describe a hysteresis loop. S* is especially important and high values are obtainable with thin-film media. S* is a measure of how steeply the hysteresis loop rises through the coercivity (H_c). The film-disk value of 0.95 is near the maximum S* value of 1.0 and is significantly improved over the particulate value of 0.68. High S* values indicate the magnetic medium's ability to switch over a narrow range of field values, which is necessary to pack bits closely together for high recording densities.

Also note in Figure 99 that the H_c is higher than that of the particulate disk and is approximately 850 Oersteds (Oe). Current IBM thin-film disk designs cover a coercivity range from 800 to 1300 Oe. Particulate disks consist of small magnetic particles dispersed in polymer resins; the particles control magnetic properties and have limitations on their coercivity range. Thin-film disk technology allows easy control of the coercivity by adjusting film compositions and sputtering conditions. IBM thin-film disks are designed to have as high a coercivity as possible within the write capabilities of the head. Future thin-film disks will have even higher coercivity values to achieve the high storage densities envisioned for future disk storage products.

The advantage of thin-film disks is further shown in Figure 99 by the comparison of the remanent magnetization (M_r) values. The M_r value is a measure of the magnetic strength of the material and governs how much signal the disk is able to generate. Thin-film disks have much higher M_r values than their particulate counterparts because no dilution of magnetism by nonmagnetic polymer binders occurs. Additionally, thin films are naturally stronger magnetic materials than their oxide counterparts (used in particulate films). Because of this 6-to-10 times higher M_r value, thin-film disks can be made thinner than particulate disks and still maintain equivalent signal output. The thinner layers allow the recorded bits to be placed closer together, yielding higher storage densities. This combination of high magnetization with low thickness is the essence of the performance advantage of the thin-film disk.

Because thin films have such high magnetization values, a process is required to precisely apply the magnetic film to the disk substrate and at low thicknesses. Magnetron sputtering vacuum equipment rapidly deposits thin films with precision and uniformity. IBM thin-film disks have film thicknesses from 500 to 700 Å and are controlled to a thickness of ±70 Å.

Of the many potential cobalt magnetic-alloy systems available, CoPtCr was chosen for the magnetic layer in IBM thin-film disks for several reasons [6]. First, the higher coercivity requirement in the design is easily achieved with this alloy, and any coercivity changes necessary for future disk designs can be attained by making slight composition changes. Secondly, the M_r value in CoPtCr is sufficiently high to produce a strong signal at low thicknesses. Third, the CoPtCr formulation gives a magnetic film with a high signal-to-noise ratio. Finally, the Cr, in addition to aiding in the control of M_r and H_c values, also serves as a corrosion inhibitor, allowing the disk to withstand extremes in temperature and humidity and to withstand the presence of foreign corrosive gases.

Each IBM thin-film disk undergoes a dynamic magnetic test of parameters, such as signal strength, resolution, signal-to-noise, and missing-bit and extra-bit defect counts. Disk drive design criteria defines the disk signal requirements, which are related to the fundamental magnetic description of the CoPtCr film. The magnetic defects are typically about one in 1,000,000 bytes and are maintained at this low level as a result of high-quality disk substrates, sputtering conditions with minimal contamination, and a series of cleaning steps performed during manufacturing that eliminate environmental and process debris.

Mechanical Durability

Thin films, even though they rest on an extremely hard surface, cannot withstand the sliding of a recording head during normal operations of the disk unit. Because IBM imposes strict standards on the durability of its recording components to ensure long disk life, designs to meet these standards were implemented on IBM thin-film disks. The sputtered amorphous carbon overcoat of 275-Å thickness protects the magnetic layer by providing a sufficiently hard surface to eliminate wear of the magnetic layer. A thin lubricant film applied to the carbon overcoat minimizes wear of the carbon film.

This lubricant minimizes long-term wear by minimizing both static and dynamic friction. Overcoming high static friction requires excessive starting torque, and high dynamic friction wears the surfaces excessively. Figure 100 shows friction traces of a thin-film disk with

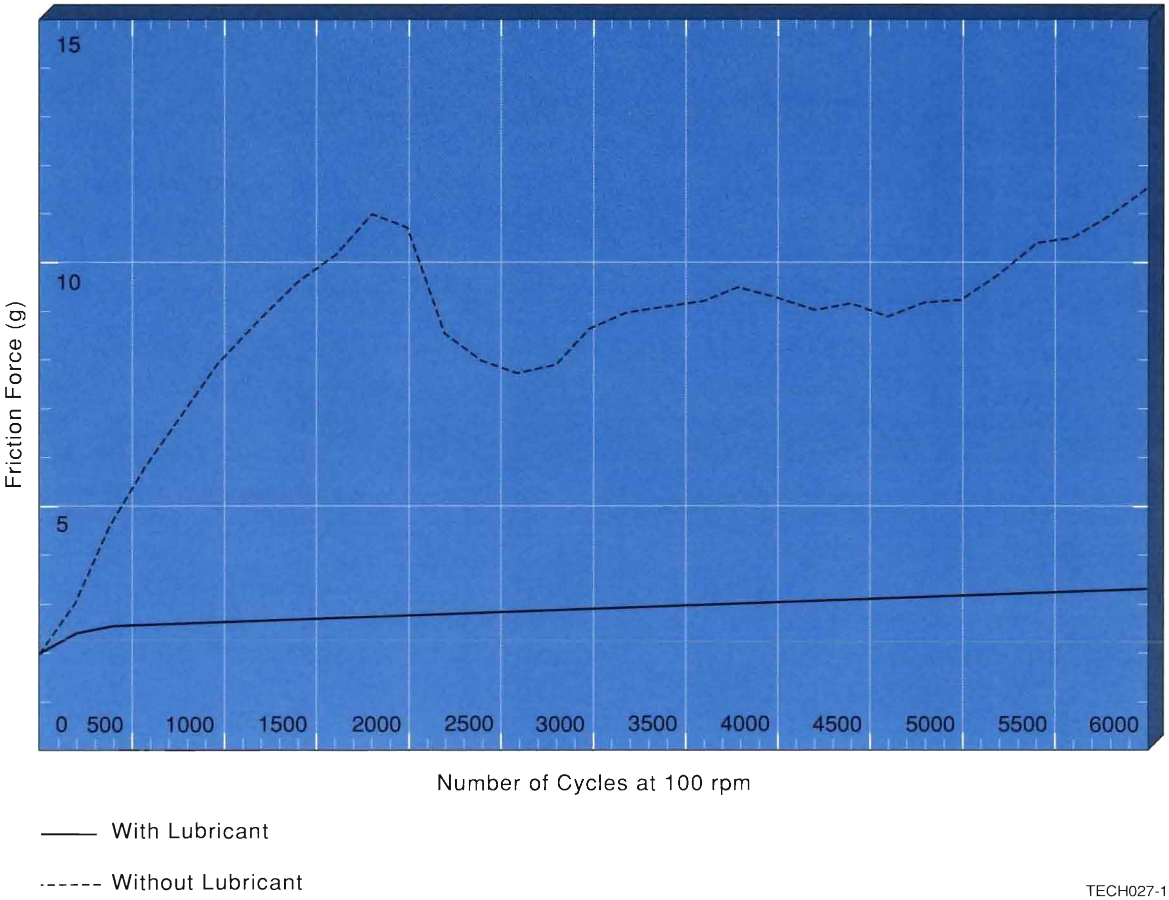


Figure 100. *Friction Force versus Number of Cycles on a Thin-Film Disk with and without Applied Lubricant*

and without an applied lubricant overcoat. The carbon surface with no lubricant has initial friction values that are low. However, after some time passes in this accelerated test, friction increases to a dangerously high value. The trace with the organic lubricant applied is extremely flat and low in value [7].

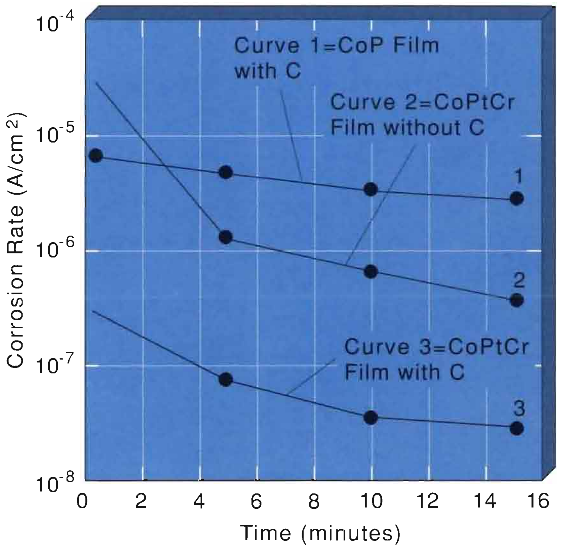
The amount of lubricant applied to the carbon surface is critical. Too little can lead to excessive wear, while too much can lead to large static friction coefficients (stiction). The optimal value is about one monolayer.

Multiple mechanical durability tests to simulate disk drive conditions were conducted. Start and stop tests mimic the take off and landing of the head on the disk surface. Flyability tests investigate the effects of intermittent contact of a flying head with disk asperities. Friction and stiction testing ensures that the head-to-disk interface is not prone to excessive drag forces. IBM thin-film disks perform these mechanical tests exceptionally well.

Resistance to Corrosion

Thin films can react chemically with a variety of materials in the environment. A major engineering effort was applied to stabilize the thin-film disks from environmental degradation. By greatly restricting air flow and using filters, the IBM disk drive design minimizes contact by airborne corrosive elements, such as chlorine and sulfur compounds, that can occur in minute quantities in disk unit operating environments. High humidity, particularly at high temperatures, then becomes the more severe corrosive exposure [8]. To alleviate surface degradation from water condensation and temperature and humidity extremes, the use of Cr in the magnetic layer is crucial. Cr readily forms a thin, nonmagnetic oxide layer, which protects the rest of the disk structure.

Film disks of other cobalt alloys, especially the cobalt-phosphorous- (CoP) plated alloys, are more reactive under stressed humidity and temperature conditions and are not acceptable for the ranges of temperature and humidity that IBM thin-film disks will experience. Figure 101 shows a corrosion-rate comparison among a plated CoP thin-film disk with a carbon overcoat (Curve 1) and IBM CoPtCr thin-film disks *with* (Curve 3) and *without* (Curve 2) a carbon overcoat. Corrosion occurs at a much greater rate on the CoP with the carbon film. Furthermore, the presence of carbon on top of the sputtered CoPtCr diminishes the corrosion rate even further. The carbon layer, although somewhat porous, serves as a protective barrier between the environment and the thin film [9].



TECH028-2

Figure 101. *Rate of Disk Corrosion in a Deionized Water Droplet*

Conclusion

The benefits of recording on thin-film disks are realized from two innovations in thin-film disk technology: improved mechanical durability from innovations in surface lubrication and improved resistance to corrosion because of the high-chromium-content magnetic alloy. The versatility in modifying thin-film magnetic properties, specifically the CoPtCr film system, makes the design of a film disk for higher densities practical. Developmental work in improved magnetic alloys will continue, with an increased emphasis on low-noise film compositions. Overcoat and lubricant technology for film disks is still in its infancy, and new discoveries are expected that will prolong disk durability over today's thin-film disks. Developments are expected that will minimize corrosion and extend mechanical life beyond today's goals. The advent of thin-film disks within IBM and the magnetic-media industry in the 1980s, coupled with other advances in storage device technology, ensure the continuation of exponential increases of storage densities in the 1990s.

Acknowledgments

IBM thin-film disks were developed at the Rochester, MN, site and are the result of a collaboration of many talented people in the disk manufacturing and development group. These disks are now manufactured at IBM sites in San Jose, CA, and Mainz, Germany, as well as in Rochester, MN. Important support work has also been done in IBM San Jose, CA, and the IBM Research Division located in Yorktown Heights, NY, and Almaden, CA.

References

1. Bate, G., *Ferromagnetic Materials*, edited by E.P. Wohlfarth, Volume 2, Amsterdam, N. Holland Publ., 381–507. 1980.

2. Arnoldussen, T.C., "Thin Film Recording Media," *Proceedings of the IEEE*, Volume 74, 1526–1591. 1986.

3. Haines, W.G., "Anisotropy in Thin-Film Media-Origins and Applications," *Journal of Applied Physics*, Volume 61, 3497–3502. 1987.

4. Howard, J.K., Ahlert, R., and Lim, G., "The Effect of Polycrystalline Sublayer Films on the Magnetic and Structural Properties of CoCr Films," *Journal of Applied Physics*, Volume 61, 3834–3836. 1987.

5. Ishikawa, M., Tani, N., Yamada, T., Ota, Y., Nakamura, K., and Itoh, A., "Film Structure and Magnetic Properties of CoNiCr/Cr Sputtered Thin Films," *IEEE Transactions on Magnetics*, Volume MAG-22, 573–576. 1986.

6. Howard, J. K., Ahlert, R., Lim, G., and Wang, R.H., "The Magnetic and Structure Properties of CoPtCr Film Media," *IBM Research Report*, Volume RJ 5198 (53818). June, 1986.

7. Lecander, R.G., IBM Rochester, MN, Private Communication.

8. Dubin, R.R., Winn, K.D., Davis, L.P., and Cutler, R.A., "Degradation of Co-Based Thin-Film Materials in Selected Corrosive Environments," *Journal of Applied Physics*, Volume 53, 2579–2581. 1982.

9. Brusic, V., IBM Research, Yorktown Heights, NY, Private Communication.

System Power Control Network

Describes how distributed power, battery backup, and a network of distributed intelligent power controllers improve system availability and serviceability.

Neil C. Berglund

Introduction

Version 2 of the AS/400 9406 system contains significant improvements in the power system to increase system availability and serviceability. These improvements are achieved using a distributed power system, integrated battery backup, and intelligent power control. Distributed power and battery backup were first introduced in the AS/400 9404 system. In the 9406, these concepts are enhanced with the addition of intelligent power control.

Intelligent power control is included in the 9406 System Unit and in each System Unit Expansion, Bus Extension Unit, and 9309 Rack Enclosure. An intelligent power control network, the **system power control network (SPCN)**, connects the operating system with the power controllers in these system resources to provide improved fault reporting, error recovery, and fault tolerance for utility interruptions and hardware failures.

Distributed Power

Distributed power uses a structure that separates power supplies into two stages. The first stage converts utility power to 28 V dc, which is distributed to the second stage and directly to loads requiring 28 V. The second stage consists of one or more regulators to convert 28 V dc to the unique voltages required by each load, that is, the various electronics and devices in the unit.

Distributed power was introduced into the 9406 to achieve and expand the advantages realized in the 9404 [1]. One advantage is the ability to use multiple, identical 28-V power supplies, operating in parallel, to provide and distribute a single voltage, nominally 28 V dc. The unique voltages and currents required by logic and devices are provided by regulators, which are packaged with, or close to, the loads they power to provide simpler distribution and tighter regulation. Supplying the 28 V dc with multiple, smaller 28-V supplies sharing the load allows power supplies with different power capacities to be built by using different numbers of the basic 28-V supply. This also allows extra capacity to be added to the total to provide n+1 capability (the ability to tolerate the failure of one supply if the remaining supplies are able to provide sufficient current).

Battery Backup

To provide the ability to tolerate most utility interruptions and selected hardware failures, a Battery Power Unit is standard in all critical system resources. The battery is turned on to power the processor and other critical resources when normal power fails. This significantly reduces the recovery time that would otherwise be necessary after the failure is corrected or power is restored.

This concept, first used on the 9404 and improved for the 9406, makes effective use of the distributed power structure. A 24-V battery backup unit is connected to the 28-V power bus in parallel with the 28-V power supplies. The batteries are turned on to provide power to the regulators if the 28-V bulk supplies fail to provide sufficient power. The batteries are turned on when utility power is interrupted and also when a hardware failure causes the 28-V power to fail. If the power failure persists, the battery provides sufficient power until the contents of main storage can be saved to a file in the system unit.

Intelligent Power Control

Microprocessor-based power controllers have been added to critical system components to make optimal use of the capabilities provided by the distributed power structure and integrated battery backup. Microprocessor-controlled power exists in the system unit and in each system unit expansion, bus extension unit, and rack enclosure.

Figure 102 shows an exploded view of significant features of the power system of the system unit. The power systems of the system unit expansion and the bus extension units are identical except they contain a simplified display panel. Utility power enters the ac module and is distributed to the three 28-V power supplies (dc bulk modules) shown across the back. The 28 V dc power is distributed from the dc bulk modules to the card enclosure where book regulators generate the voltages for the system processor, main storage, and I/O controller cards. The integrated load source file, which is packaged within a book and located in the card enclosure, receives 28 V directly. The load source file regulator is contained within the file book package. Also connected to the 28-V bus is the Battery Power Unit, located above the card enclosure. The power controller is located in the ac module. Status and control signals connect the 28-V power supplies, the battery, and each regulator to the power controller, which is also connected to the control panel shown directly in front of the battery.

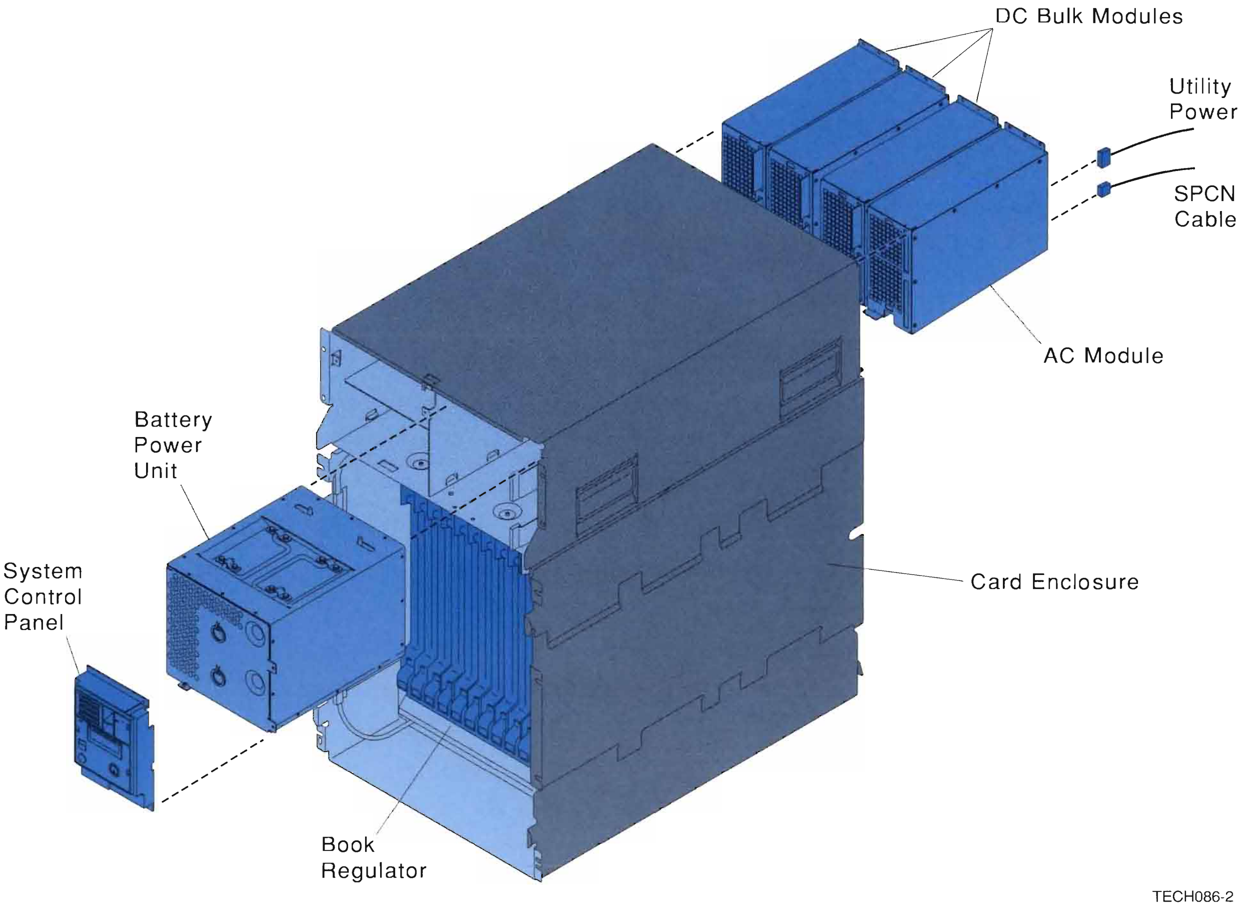


Figure 102. *9406 System Unit Power Control*

The power controller turns power on and off in the unit and can individually control each regulator. When power is turned on, the controller constantly monitors the battery, each 28-V supply, each regulator, and the cooling blower. The power controller exists to detect and report power system malfunctions and to keep the unit operating, if possible, in spite of the malfunctions. When a utility interruption occurs, the 28-V power supplies cease to provide sufficient power. The battery is turned on to keep the unit operating and the power controller notifies the operating system that battery power is active. The power controller also notifies the operating system when utility power is restored. If utility power is not restored within a sufficient time, the operating system saves main storage to a file in the system unit and powers the system off. When utility power is restored, the system automatically powers on, reloads main storage from the file, and performs the necessary recovery.

To provide additional fault tolerance, the multiple 28-V power supplies and the battery work together to back up hardware failures. If one 28-V supply fails to provide sufficient power, the others attempt to share the increased load. The system continues to operate normally while the power controller reports the failing component to the operating system. If the system load exceeds what can be provided by the power supplies that remain operating, the battery is turned on to power the unit. The power controller warns the operating system that battery power is active; the operating system saves main storage to the file and powers off the system for repair.

To ensure that the backup power systems are working, the power controller detects the presence of the battery unit and provides continuous monitoring and various test capabilities. Battery failure and end of life are among those conditions that are detected and reported to the operating system to warn the user of the loss of backup capability.

System Power Control Network

To make optimal use of the intelligent power functions in the critical system components, the power controllers in each component are interconnected with a serial communications network. The system power control network is a hierarchical intelligent communications path that connects the operating system and the power system in components containing intelligent power control. SPCN provides the ability to report power status, power supply and battery failures, utility interruptions, and configuration changes from any element in the network directly to the operating system.

The SPCN network consists of a path from the operating system to the system unit power supply, from the system unit power supply to the racks containing additional system resources, and from the racks to the individual system components (units) within each rack. The system unit, system unit expansion, and bus extension unit are connected to the SPCN power controller in the rack using short copper cables. The racks are serially interconnected using copper cables or optical fiber to achieve noise immunity and physical installation flexibility previously unavailable. Figure 103 shows the interconnection of the power controllers in the system units and racks.

The interface between the operating system and the serial power network is provided by the power controller in the system unit power supply. In addition to monitoring and controlling system unit power supplies and the battery, the power controller constantly monitors the status from all SPCN controllers in the network and reports critical changes and faults to the operating system. The system unit power controller also receives commands from

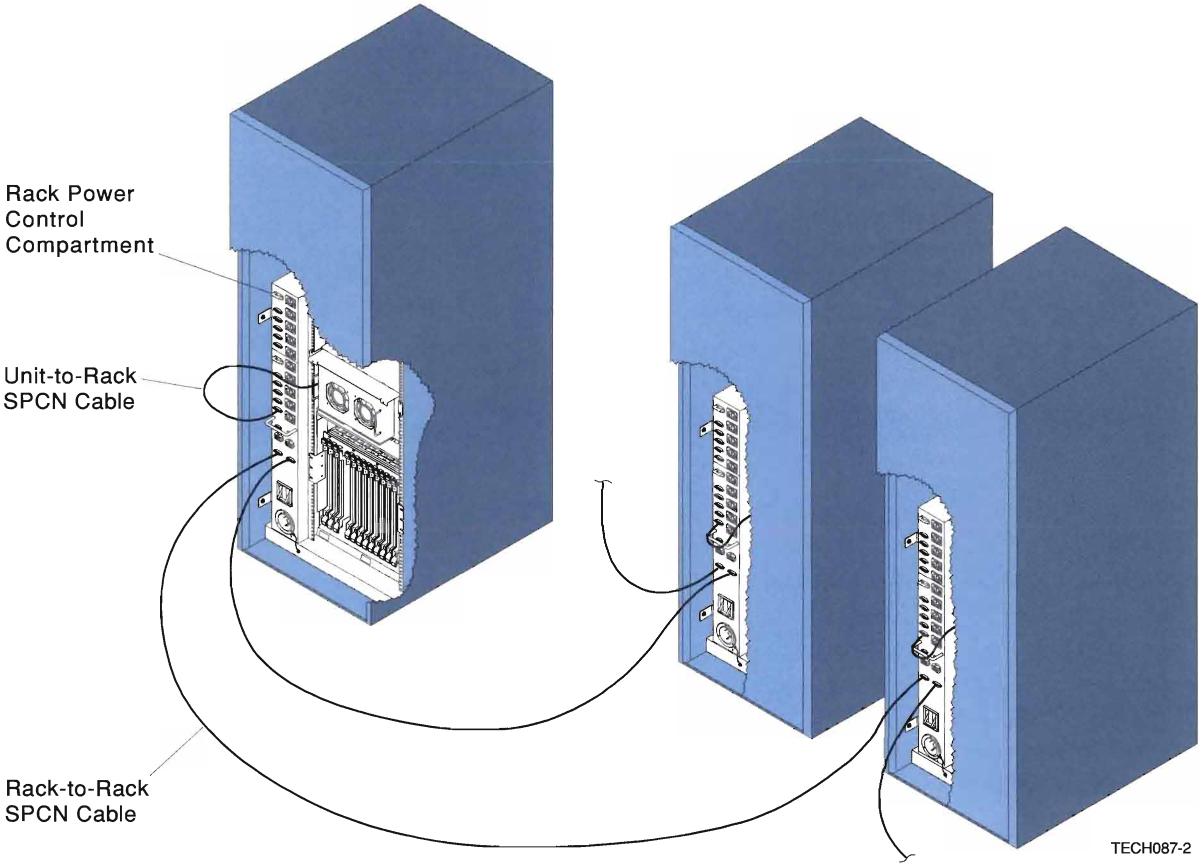


Figure 103. *System Power Control Network*

the operating system and either processes them directly or forwards them to the addressed controller in the network.

The rack power controller provides control of utility power for system components in the rack and serves as the router of serial network traffic. Commands from the system unit controller are received by the rack and forwarded to a unit in the rack or to another rack as directed by the command address. Responses are returned to the system unit power controller and then to the operating system.

Each power controller in the network monitors and controls the power supplies and the battery within its respective unit and reports critical state changes and faults to the system unit controller in response to periodic network polling commands. Faults and battery status are also displayed on the local display panel attached to each controller.

Impending Power Failure

An interconnection between intelligent power controllers and the operating system is essential to system availability because the operating system must prepare for a safe shutdown when power interruptions or failures threaten the system unit or any unit containing disk I/O processors. Because utility interruptions or power supply failures may only affect a part of the system, SPCN alerts the operating system when any unit in the network warns of impending power failure. If normal power is not restored, the operating system saves main storage on a file before battery power is exhausted.

Customers may choose to provide an **uninterruptible power supply (UPS)** to provide additional backup capacity or to back up those parts of the system not protected by the AS/400 internal batteries. To provide operating system notification of utility failures reported by a UPS, each rack contains an interface to receive UPS status. UPS status may be connected to any rack, and, if multiple backup sources are used for a multiple rack system, status from each UPS may be connected to a different rack. UPS status from each rack is collected and reported to the operating system, using the network, in the same manner that internal battery status is reported. The internal battery is not used to back up utility failures that are handled by an attached UPS. However, the internal battery remains enabled to back up hardware failures and to provide backup if the UPS is offline, insufficiently charged, or failing.

Power Fault Reporting

In addition to warning the operating system of impending power failures, SPCN also serves as the means to identify and report the root cause of problems that would otherwise be unknown or indistinguishable at the operating system interface. The set of faults that cause power to be removed from the critical system resources, such as the system processor, I/O processors, or device clusters, usually disables the functional reporting paths between the failing unit and the operating system. Consequently, power problems appear to be problems in other components. SPCN distinguishes power failures and user-caused problems from functional problems, and reports the problems to the operating system so that IBM service can be dispatched with the correct parts.

The addition of integrated backup capability for hardware and utility failures creates an additional need for automatic fault reporting. For example, the failure of one of the 28-V power supplies or of the battery does not impair system operation until the failed component is needed. SPCN provides the means to report this failure; the user is warned that backup capability is disabled, and service can be dispatched while the system continues to operate.

Conclusion

A distributed power structure, battery backup, and intelligent power control are introduced to the 9406 to improve system availability and serviceability. Availability is improved with fault tolerance for hardware and environmental problems. Availability and serviceability are improved through the discrimination and reporting of problems caused by power failures. These capabilities are integrated through a power control network that provides uniform access to power control functions from the operating system.

Reference

1. Squillace, Z.D., Tenley, R.A., Lukes, F.J., and Reckinger, A.P., "Power, Packaging, and Cooling for the 9404 System Unit," *IBM Application System/400 Technology*, SA21-9540, 142–144. June, 1988.

Electromagnetic Compatibility Design

Describes the electromagnetic compatibility (EMC) design of the Version 2 AS/400 system and its importance in increasing system reliability and availability through increased noise immunity.

Kevin J. Przybylski and Scott M. Thorvilson

Introduction

Electromagnetic compatibility (EMC) is a central design goal of the Version 2 Release 1 AS/400 system. The major objectives of the EMC design are to:

- Prevent external electrical noise from causing operational problems and malfunctions
- Prevent system emissions of electromagnetic interference (EMI)

The overriding goal of the EMC design is to ensure proper system operation in any customer environment. This required that the AS/400 9406 System Unit be impervious to all forms of conducted and radiated noise that may be found in a system installation. To accomplish this, the 9406 is designed for and tested to the highest standards of noise immunity in four key areas:[11]

- Electrostatic discharge (ESD): The system is designed to be impervious to the high-voltage, high-current discharges that result from contact between the system and charged furniture or personnel.
- Radiated electromagnetic susceptibility: The system will operate properly in areas with strong radio frequency interference from nearby television, radio, microwave, or radar installations.
- Lightning susceptibility and power line disturbances: The system will maintain proper operation during lightning and power-line disturbances, including lightning strikes near power lines or external cables, surges and sags due to power company switching, brownouts, and even complete loss of power.
- Electromagnetic interference: The system is designed not to be a source of interference to other equipment in the customer's installation.

The high level of EMC performance required by the system is achieved by an integrated design approach. Proven EMC designs are incorporated into every major system component including licensed internal code, electronics, power systems, and mechanical packaging.

EMC Design

The EMC design of the 9406 consists of three major elements: the electronics design, the power systems design, and the overall mechanical packaging design. The objectives of the EMC design are to harden the internal electronics against noise, to provide power-line noise immunity through power filtering and battery backup, and to provide high performance shielding protection against all external noise threats by using a new mechanical design based on the advanced packaging introduced in the AS/400 9404 System Unit.

Electronics Design

The advanced electronic function and the speed of the 9406 require that EMC guidelines and techniques be a basic part of all the card designs. Special attention is devoted to ensuring that all of the circuit boards are designed and laid out for optimal noise immunity, and that each board incorporates noise suppression components into critical circuitry.

All electronic circuit boards within the system use advanced multilayer card designs for maximum operational and EMC performance. These designs combine four to six signal

[11] The 9406 exceeds current American National Standards Institute (ANSI) and Institute of Electrical and Electronics Engineers (IEEE) immunity specifications.

planes with two to four internal power planes to provide a high performance design capable of supporting the advanced function provided by the dual processor, 384MB (MB equals 1,048,576 bytes) design of the 9406. The full internal power planes of these designs provide a low-impedance return path for all signal and power currents, eliminating noise transients due to return path discontinuities. The spacing between the internal power planes is also controlled to provide an optimal level of high frequency filtering, which reduces the amount of high frequency noise on the cards and on the system backplane.

Decoupling

In addition to internal power planes, **decoupling** (power filter) capacitors are used on all of the system circuit cards to reduce the amount of high frequency noise at the circuit level. On some critical cards, customized decoupling is used to reduce a specific frequency. In these cases, a capacitor, whose resonant frequency matches the frequency of concern, is used to eliminate the on-card switching noise.

Circuit Layout

All cards are laid out with EMC design guidelines as an integral part of the process. This includes controlling component placement and orientation to minimize clock and bus lengths, and routing critical signals away from connectors and a card's edges. Other noise reduction methods include the placement of ground return lines on each side of clock and critical signal distributions, isolation of noisy signals from sensitive ones, and the use of balanced differential circuits.

Circuit Design

All cards in the system are specifically designed to improve noise immunity. Filter networks are added to the critical signal lines (such as the system reset line) on all the cards to increase their noise immunity. Other sensitive circuits, such as the phase-locked-loop system clock circuitry, have increased noise immunity by embedding critical frequency-determining components within modules and by careful use of filtered power. Special error recovery hardware and software is also included in all of the logic designs to retry operations and to detect and correct data that may have been affected by noise. These techniques, along with the advanced packaging design, provide improved system availability and ensure data integrity.

Packaging Design

In conjunction with the advanced design techniques used for noise reduction and immunity on all the electronics, a new packaging strategy is used to give the system the best possible shielding protection available. This new mechanical system design is composed of the book package and the card enclosure, the backplane and docking assembly, and the power support box. The system is structured such that electronic cards are packaged within the book, the books are installed within the card enclosure, which in turn is docked with the power support box. The combination of all of the above elements is known as the processing unit and is shown in Figure 104. The processing unit is 0.80 m high, and mounts in the current 1.6 m 9309 Rack Enclosure.

Processing Unit

The heart of the system resides in the processing unit. The EMC strategy for this design is to package all of the critical processing unit components at the unit level. These elements include the system backplane, the system processor, main storage and input/output cards, and the power regulators.

The basic building block used to provide all of the unit level protection is the mechanical book

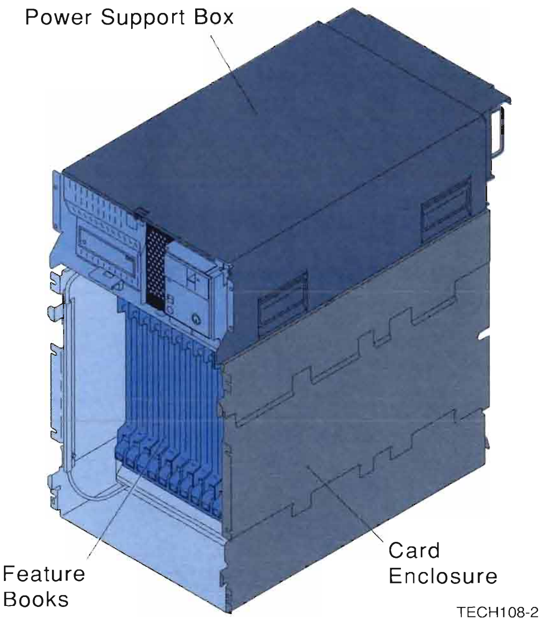


Figure 104. *Processing Unit.* The card enclosure and the power support box are docked together to form the processing unit.

package. The book packaging concept is used to create a modular system that allows the customer the freedom to create a custom system configuration. This allows the number of main storage cards, work stations, communications lines, and disk I/O capacity to be controlled by the simple addition of the proper card and book assemblies.

The book itself consists of two die-cast magnesium-alloy covers and a tailstock, which are used to completely shield each electronic assembly, as shown in Figure 105. The book is grounded to the castings of the card enclosure by spring fingers attached to the tailstock. The tailstock is custom designed for each type of card to provide proper grounding of the book and the attached cables, and to provide shielding for each card. This is most apparent in the book that houses the optical interface card, which uses a tailstock that extends well beyond the book to protect the sensitive optical transmitter and receiver as shown in Figure 105. All tailstocks use a special copper-nickel alloy for its high conductivity and optimal grounding capabilities. A book can be inserted into any slot in the card enclosure with this package. In the absence of a card, a filler book is used to help provide proper air flow and EMC protection for the exposed backplane and connectors.

Another important benefit that the book package provides is handling protection. The book in combination with circuitry on the card provides unsurpassed static charge dissipation thus protecting the sensitive electronic components on the card from damage due to ESD. The book delivers this protection by fully enclosing all of a card's components in the highly conductive book package. The book also provides a resistive logic ground connection to safely dissipate static charge before the card is plugged into its backplane connector. The rigidity of the book package also protects the card from physical damage.

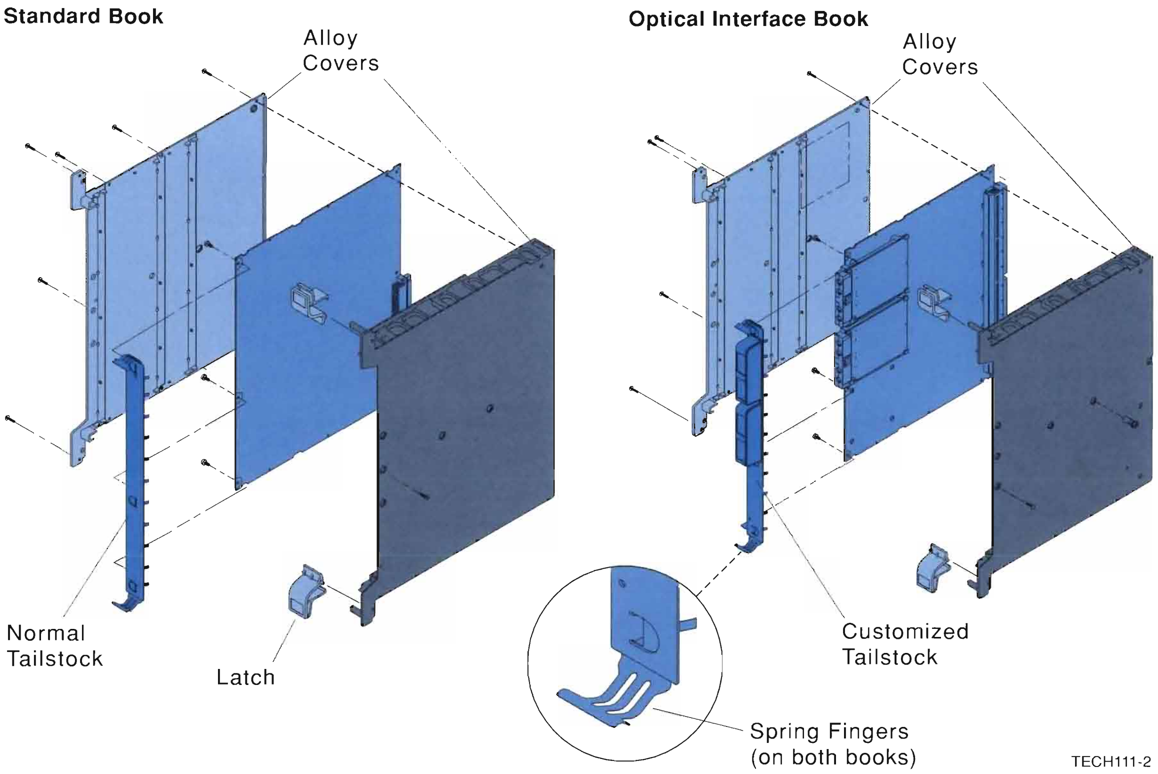


Figure 105. *Book Package*. Generic books are shown with both a normal tailstock and a specially modified optical card tailstock.

All modular books plug into the card enclosure as shown in Figure 106 on page 201. When inserted, the cards are connected together by the backplane. The backplane is protected by shields in the front and back as shown in Figure 107 on page 201. The backplane shields are bolted to the castings and the side plates of the card enclosure to complete the shielding of the sides and bottom of the backplane. The backplane shield covers all parts of the backplane except the card connectors. Complete shielding of the backplane is accomplished when all the books are inserted into

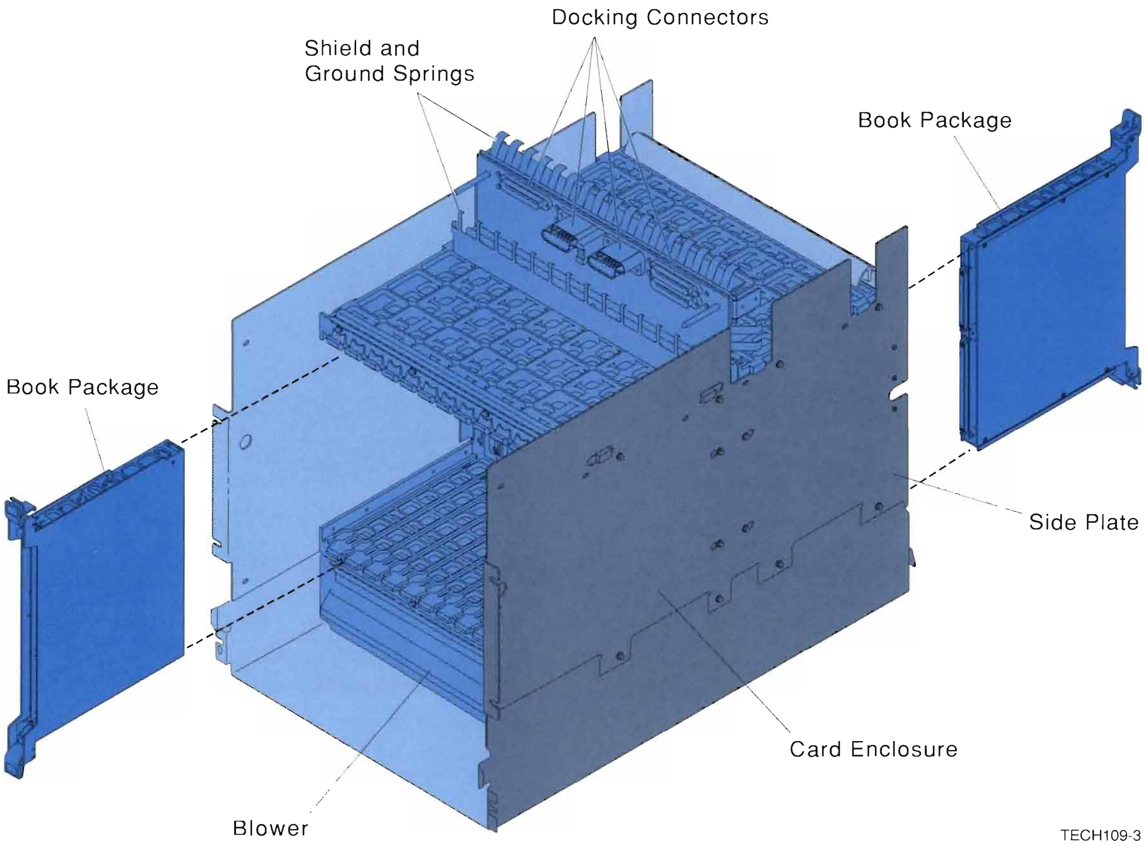


Figure 106. *Card Enclosure*. This enclosure provides shielding and grounding for all the books, the backplane, and the docking interface.

the card enclosure, each making contact to the backplane shield through spring contacts on the shield shown in Figure 107. With all books in place, a complete shield is formed around the cards and the backplane.

Docking

Because of manufacturing and service needs for quick assembly and disassembly, a plug-in style of docking is implemented for connecting the power support box to the card enclosure.

As shown in Figure 106, shields and ground springs are used to shield the exposed backplane, power distribution, and control panel connectors at this interface. This shielding is designed to bridge the gap between the power support box and card enclosure, completely sealing off exposed areas and providing a solid ground structure between the card enclosure and the power support box.

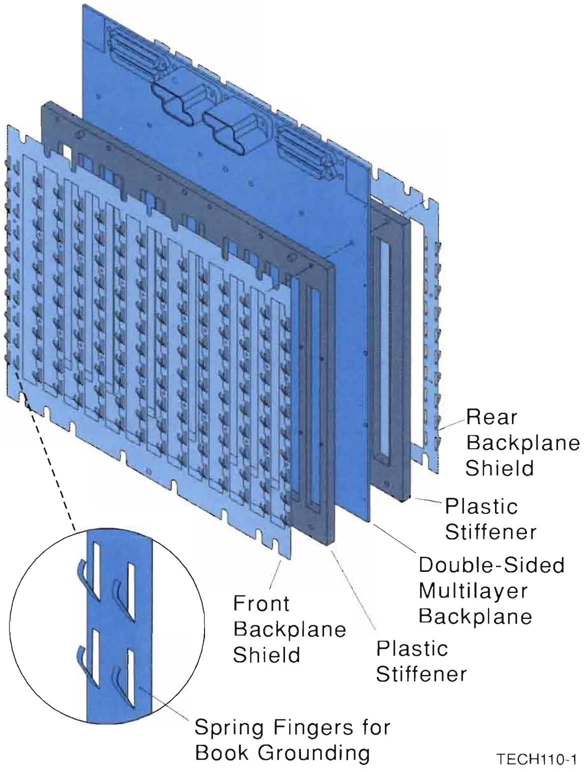


Figure 107. *Backplane Shield*. This shield uses spring fingers to make contact with books, providing a continuous shield on both sides of the backplane.

9406 System Mechanical Packaging

Describes the mechanical packaging design of the 9406 System Unit.

Michael D. Seyfert, Zanti D. Squillace, and Mark M. Thornton

Introduction

The mechanical packaging of the Version 2 Release 1 9406 System Unit provides for the physical, thermal, and electromagnetic protection of the logic, power, battery backup, save and restore device, and integrated disk drive (see Figure 108).

This processing unit package consists of two major units: the card enclosure and the power support box (see Figure 109). These units are used together in the 9309 Rack Enclosure. The rack contains power input lines plus power control devices. When the power support box is mounted in the rack, it is located just above the card enclosure. These two units are joined through a unique device that completes the electrical interconnections required for power and control circuits.

The processing unit hardware is reused in the System Unit Expansion and the Bus Extension Unit. These units allow for additional system input/output (I/O) functions.

The processing unit package was designed to support the performance and price objectives of the total system. Performance requirements drove the need for a new processing unit package to accommodate the higher processor speeds and the resultant increase in radio frequency emissions.

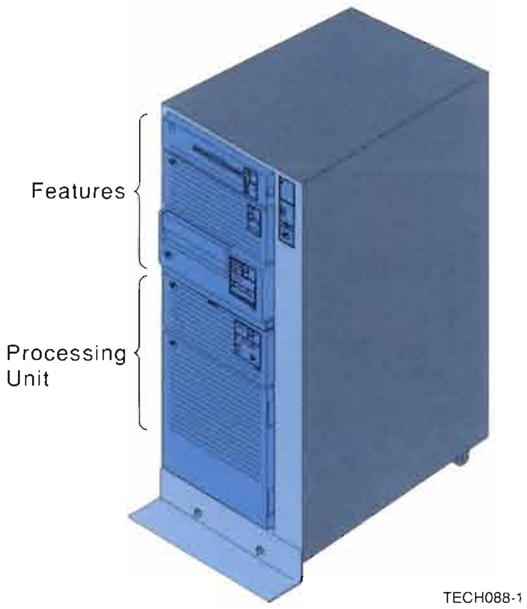


Figure 108. *9406 Processing Unit*

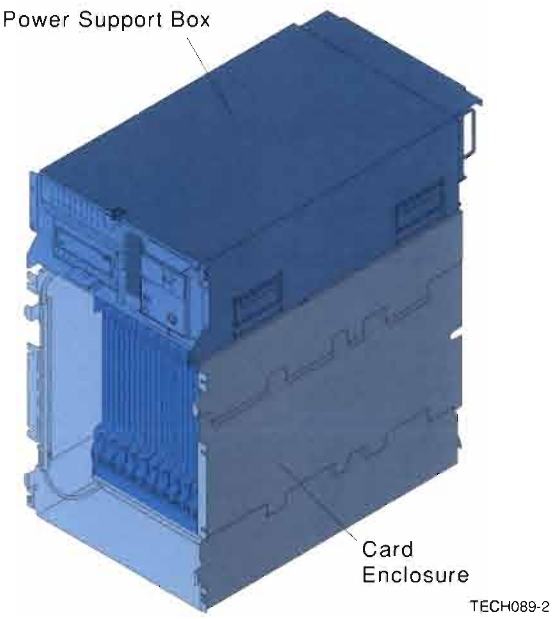


Figure 109. *Mechanical Package for the Processing Unit*

The processing unit package includes modular power boxes, a save and restore device, a battery backup unit, and book packages. Book packages are individual containers that house:

- Integrated disk units
- Processor cards
- Main storage cards
- I/O cards
- I/O bus expansion cards
- Voltage regulator cards

Card Enclosure

The **card enclosure** (see Figure 110) is an assembly that provides support and interconnection of the book packages to the main backpanel. Several types of book packages are used to support cards needed in the system. Also, a special book was designed to integrate the 3-1/2 inch disk units within the processing unit as a base item.

The heart of the card enclosure is the unique double-sided backpanel. The backpanel uses compliant-pin connectors. A **compliant-pin connector** has a pin design that makes a press-fit connection to the backpanel. The advantage of compliant pin connectors is that it allows logic cards to be plugged into the backpanel from both sides. The average card-to-card wiring length decreases by a factor of two, which provides increased performance and function. Another benefit with the reduction of internal cabling is improved reliability. The backpanels for the various card enclosures are different. There are four backpanel designs, one for Models D35 and D45, Models D50 through D80, the system unit expansion, and the bus extension unit. While the backpanel designs vary among the card enclosures, the surrounding mechanical hardware is common. Common hardware reduces tooling costs, and decreases parts costs through increased part volumes.

The backpanel is sandwiched between molded stiffeners and stainless steel shields (see Figure 111 on page 206). The stiffeners minimize backpanel movement during insertion and withdrawal of the book-packaged elements.

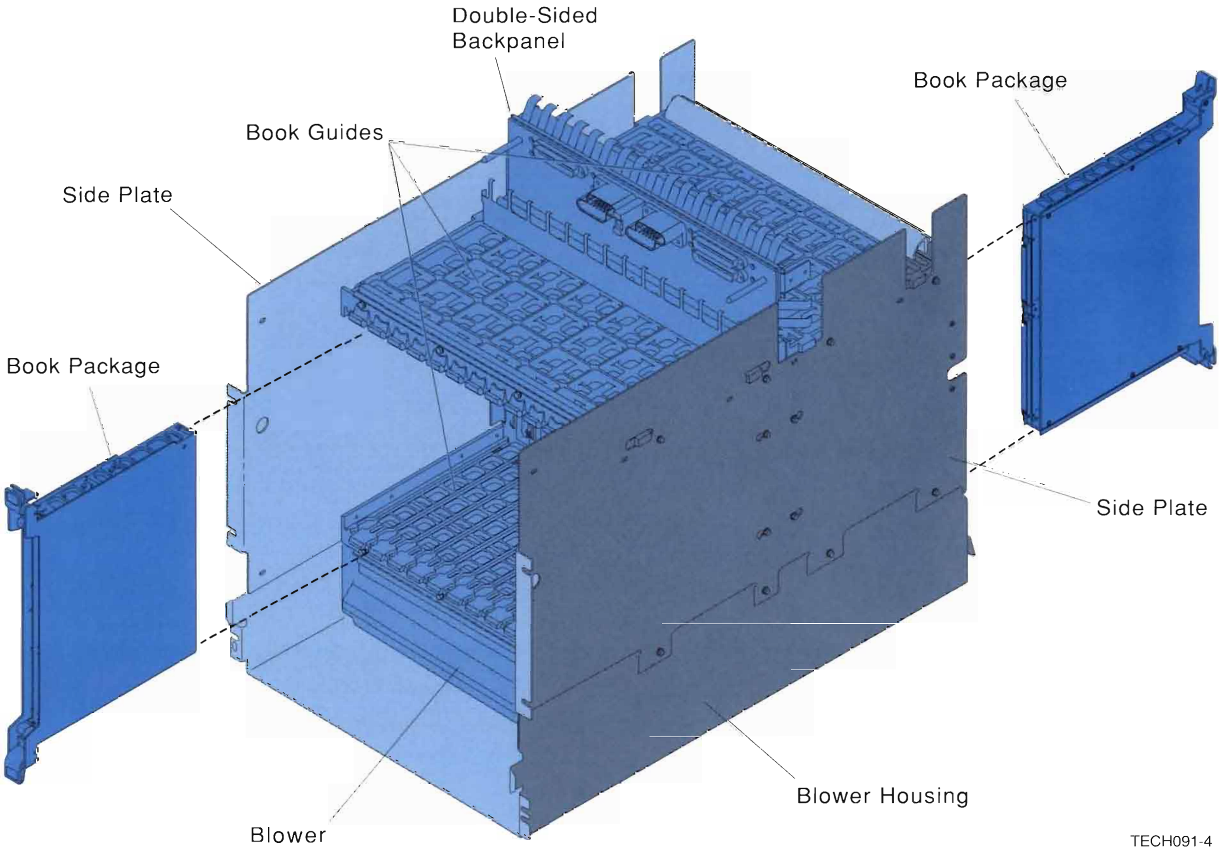


Figure 110. *Card Enclosure*

The shields provide radio frequency interference (RFI) shielding of the backpanel while providing book-to-book grounding.

The stiffeners were extensively analyzed using a computer simulation technique called finite element modeling. This modeling allowed the design to achieve the desired stiffness with a low-cost, commonly available plastic material. With four double-sided backpanel designs, normally eight stiffeners would be required. Using this model, the number of stiffeners was reduced to three. Finite element modeling was also used to evaluate molding characteristics of various materials to further improve stiffness.

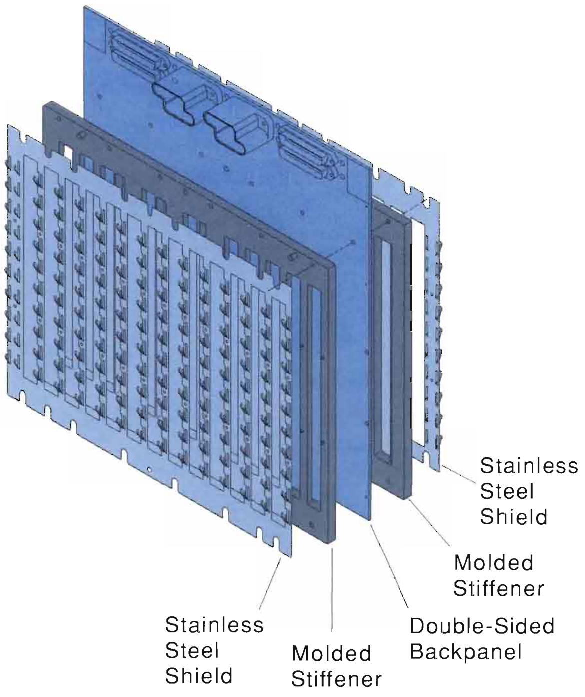


Figure 111. *Backpanel Subassembly*

The stainless steel shield is designed to provide the required RFI protection while taking advantage of symmetry so that one tool and one part can be used on the front and back of each of the four backpanel layouts.

The backpanel, stiffeners, and shields assembly is surrounded by the card enclosure assembly. The double-sided card enclosure assembly is a larger version of the single-sided card enclosure assembly used in both the AS/400 9404 System Unit and the mid-range models of the System/390 computer. The book guides provide initial connector guidance during card plugging, and provide additional RFI protection to the logic cards within the card enclosure.

Large, one-piece, side plates are designed to avoid damage from typical shock and vibration loads. These loads can come from a variety of sources, for example, shipping and relocation. With simple flanges on the side-plates, a direct attachment is made to the rack enclosure, further adding to the system rigidity.

Below the card enclosure assembly, a separate housing supports a blower and contains the airflow path. From a cooling point of view, the blower is embedded between the front and rear parts of the card enclosure. Physically, the cooling air is drawn down through the front part of the card enclosure, through the blower, then up through the rear part of the card enclosure. With the blower embedded in the cooling path, acoustic levels have been improved from the Version 1 Release 1 9406 processing unit.

The logic cards in the card enclosure are mounted within subassemblies called **book packages**. The book package primarily offers additional RFI protection to the logic cards, which is necessary as a result of the increased system performance. The book package has several advantages, such as providing physical protection for the cards while handling and dissipating electrostatic charges, which can cause latent card failures. When the card enclosure is not fully configured with book assemblies, a filler book is placed in the unused slots to ensure protection from RFI and proper cooling characteristics within the card enclosure.

The book package has been extended to hold 3-1/2 inch disk units (see Figure 112 on page 207). The Disk Drive/Adapter Assembly contains two 320MB (MB equals 1,048,576 bytes) disk units, along with the power cabling and voltage regulator assembly. The first disk assembly contains the code for the initial program load (IPL) while retaining the capacity for a main storage dump in the battery backup mode. Unused capacity is available for customer use like any other disk unit. The second disk assembly provides additional capacity for customer use or mirroring for improved availability. For customer security, the integrated disk assembly contains a latch, which allows for locking capability.

Power Support Box

The **power support box** (see Figure 113 on page 208) is an assembly that contains an ac power module, three dc power modules, a 1/4 inch save and restore tape device, a dc battery backup unit, a docking tray, a control panel, a fan box, and interconnecting cables. Power is supplied to the card enclosure from the power support box. The dc power modules convert 220 V ac into 28 V dc, which is distributed throughout the system. To improve the system's availability to the customer, the dc power modules are designed for continuous system operation should one unit fail. In this design, modules share output current and are sized to provide sufficient power with only two of the three modules running. The ac module package houses a stage of the line filter, the

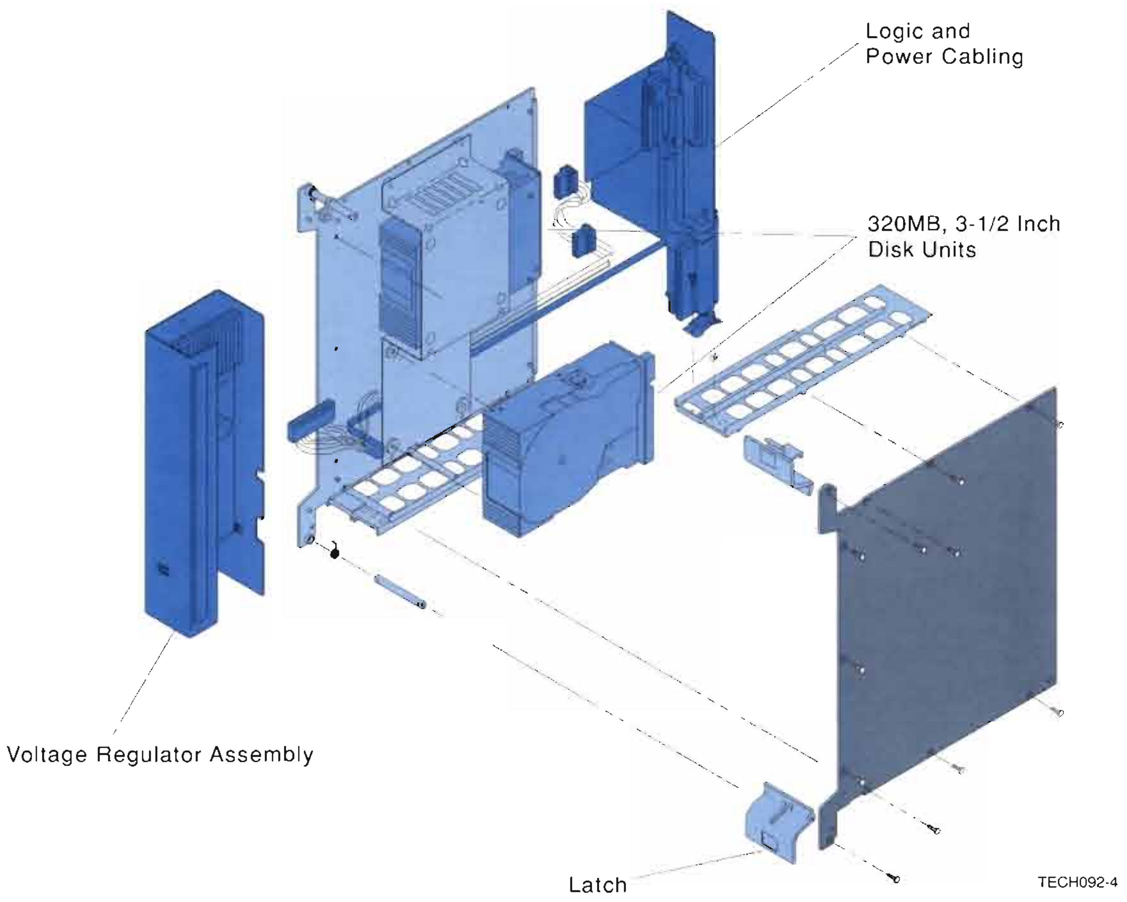


Figure 112. *Disk Assembly*

system power control network (SPCN) board assembly, and the control power supply. The control power supply enables the control panel to be active even when the power has been turned off to the remainder of the processing unit. The dc and ac power modules are designed for quick service.

The battery unit is standard with the system and provides backup power for the card enclosures to operate during most power interruptions. If the interruption persists, the system dumps main storage to the integrated hard disk drive to preserve customer data and improve IPL time. The battery unit package is designed for ease of replacement and quick service.

The power support box contains a 1/4 inch tape save and restore unit as a feature item. The tape unit is packaged in its own modular tray assembly along with a power regulator, power cables, and small computer systems interface (SCSI) bus cables. The operator accesses the tape unit by pressing the cover latch to drop down the cover on the power support box.

At the center of the power support box is an assembly called the cable carrier. Its purpose is to link power between the power support box, the tape unit, the control panel, and the card enclosure. The cable carrier contains all of the cables and connectors in a modular plastic package, which is easy to manufacture and assemble into the unit.

The power support box is electrically connected to the card enclosure with a unique hardware device called the **docking tray** The docking tray aligns the support box connectors with the backpanel connectors. This connects the SCSI bus, power, SPCN, and fan signals between the two boxes.

The power support box has its own cooling device. This fan enclosure, located at the rear of the power support box, draws air from the front of the unit, through the vents around the tape unit and the battery unit, and then through the power supplies.

The control panel (see Figure 114 on page 208) has Processor Activity lights, a Delayed Off power switch with a light emitting diode (LED) indicator, a System Attention light, a Function Select switch with Enter button, a liquid crystal display (LCD) for displaying fault codes, and a keylock. The Processor Activity lights display processor use in 10% increments for each processor. These allow for tuning the

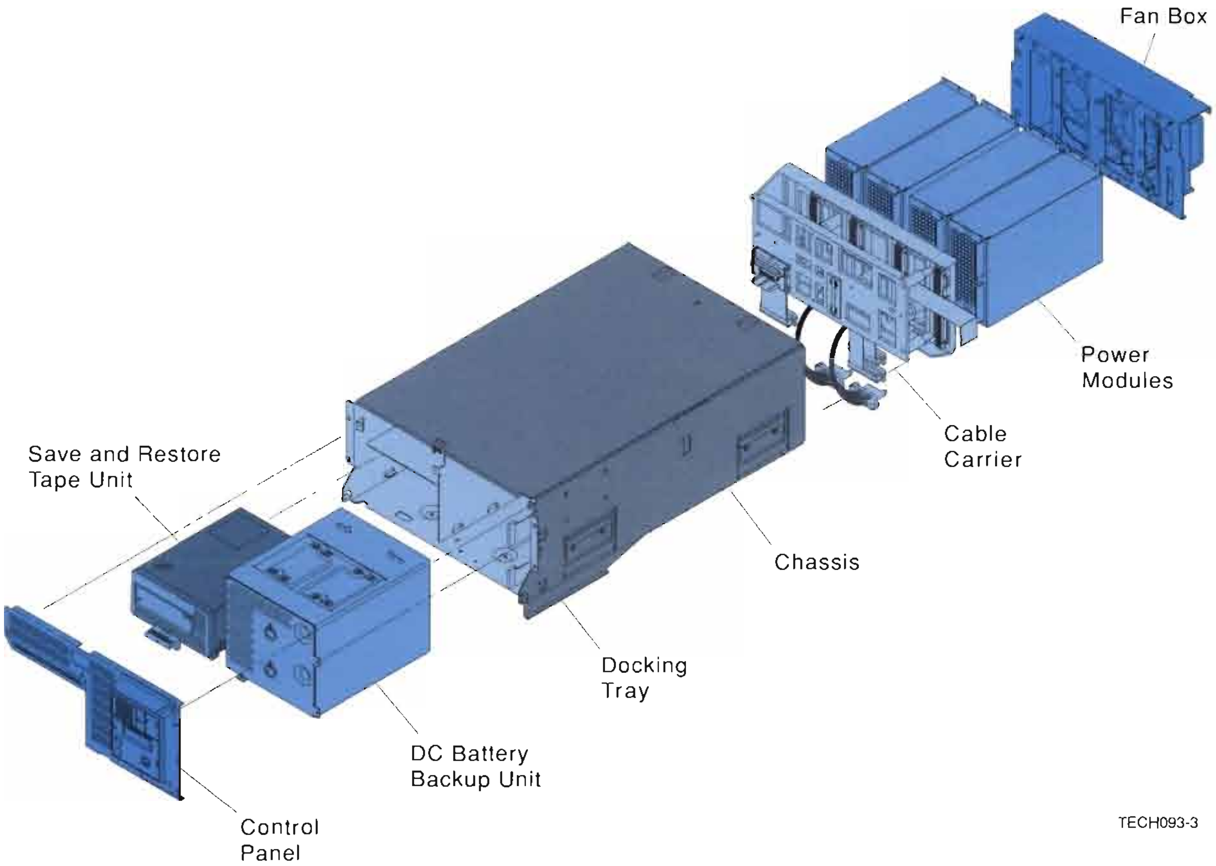


Figure 113. *Power Support Box*

code of specific applications and give a visual indication of a potentially hung system.

The control panels on the bus extension unit and system unit expansion have a Power switch with an LED indicator and an 8-character LCD to show power faults and battery status.

Hardware Package Design

The hardware package design supports the Version 2 Release 1 system capabilities and protects the customer's investment by providing modular expansion methodologies to support future system growth in power, cooling, and electromagnetic compatibility (EMC) requirements.

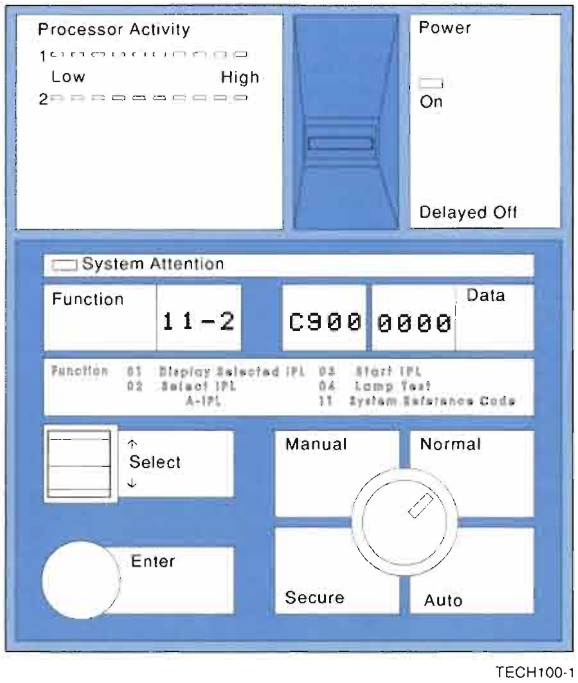


Figure 114. *Control Panel*

Conclusion

The mechanical package design is a highly functional and cost-competitive unit that meets the present requirements and provides for future growth.

Compliant-Pin Technology

Describes the high performance backplane technology.

Jeffrey A. Collett

Introduction

The AS/400 development team designed the 9406 System Unit around the High Density Plus** connector system and compliant-pin backplane technology. The solderless, high-performance backplane developed for this system allows the system to achieve enhanced packaging density with the capability for growth in performance and function.

Compliant-pin technology has been used in military products for many years and has been used in commercial products for a number of years. The compliant pin is press-fitted into a plated-through hole in the printed circuit board. Electrical contact is made and retained through a specially designed section of the pin that deforms elastically (partially) inside the plated-through hole. The large radial pressures produced in this process make a reliable gas-tight seal. Measured contact resistance in compliant pins is less than one milliohm and is stable throughout the product life. Three different types of compliant-pin designs are used in this product. One of the common design types is the eye of the needle. Figure 115 shows an eye of the needle before pin insertion. Figure 116 shows a cross-section of a plated-through hole with the pin inserted, showing the contact in the hole.

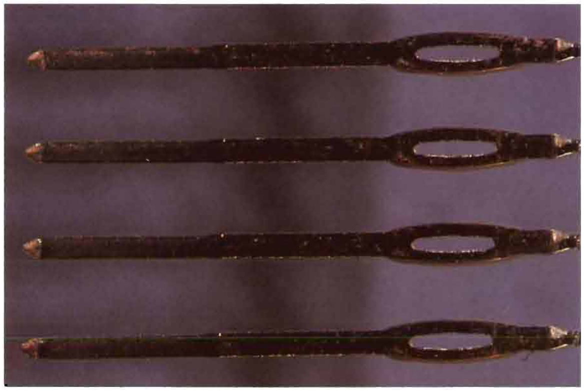


Figure 115. *Eye of Needle-Compliant Pin*

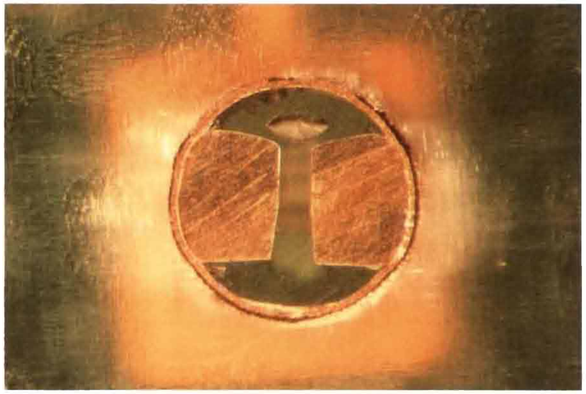


Figure 116. *Cross-Section of Plated-Through Hole with Compliant Pin Inserted*

The principal advantage of the technology is the solderless assembly process. Solder machines and washing processes are no longer necessary. Connectors are inserted into the board by a specially tooled press. By designing the appropriate supporting hardware, it is easy to produce assemblies with connectors inserted from both sides of the board. It is this feature of the assembly technology that generated interest. Most connectors are available or can be tooled in compliant-pin form so that mixed process backplanes are not necessary.

The main disadvantage to the use of the technology is the requirement for thick (>3.18 mm) backplanes to contain the stresses developed in the assembly process. Another problem is that every connector vendor tends to have unique designs for the compliant section of the pin, resulting in the need for extensive product evaluation. The design multiplicity problem was handled and the additional board thickness was used to produce an enhanced backplane wiring structure.

Implementation

The new 9406 backplane and card enclosure are extensions of the 9404 System Unit design. The 9404 backplane contains eight card slots on a 30.48-mm pitch populated with three-row connectors. This is a single-sided assembly and is held in a card enclosure that fits in a custom set of frames and covers. The 9406 is packaged in a standard 495.3-mm rack and contains more cards than could be handled by the Version 1 enclosure. For this program, the packaging concept was expanded to a double-sided backplane with front and back side connectors interleaved. This type of backplane is nearly impossible to manufacture without the use of compliant-pin technology. The enclosure width is expanded to fit the rack and now contains 13 potential card slots per side. The backplane is sandwiched between two enclosures in the processing unit. An extension above the enclosures contains compliant-pin connectors that are used for bulk power and system backup connections. The backplane has no soldered components. Although compliant-pin technology is used in other commercial products, those products have not exploited the full potential of the technology to enhance system density by introducing card enclosures on both sides of the backplane.

High Performance Connectors

The higher levels of performance needed for Version 2 of the AS/400 system could not be supported with connectors used in previous releases because of their pin count limitations. This problem is solved by the addition of the High Density Plus connector for selective use on high performance functions. This connector is a modular, four-row, shielded connector. Individual modules are 38.1-mm or 50.8-mm long and are held together by an aluminum extrusion. Optional metal shield contacts can be placed outside of one or both sides of the socket part of the connector.

Figure 117 shows the shield contacts on High Density Plus One connector modules used in Version 2 of the AS/400 system. These shields are connected to separate rows of pins that solder into the card. The shield contacts couple with a set of contacts placed in the walls of the backplane connector. The shield contacts provide a ground reference adjacent to the short and long rows of the connector. These contacts improve power distribution to the card and reduce electrical noise generated within the connector. The shield contacts outside the row of long pins are especially effective in reducing the effective inductance and coupling of the long pins in the connector. These characteristics allow the designer to devote fewer pins in the signal field to power and ground for shielding and power distribution and, thus, increase the number of pins available for signal connections. The processors shipped with Version 2 use the High Density Plus One version of the connector system.

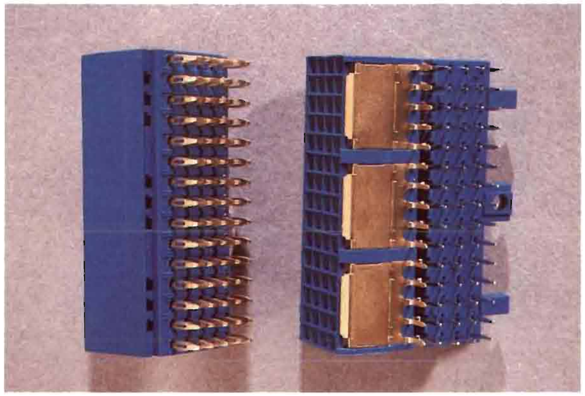


Figure 117. *High Density Plus One Connector Modules*

These performance characteristics were validated by laboratory measurements of connector inductance, capacitance, and time delay.

These measurements were used in the modeling process to guarantee signal integrity in the system. The High Density Plus connector was measured and compared with other standard four-row, right-angle connectors. When measured using a 3:1 signal-to-power ratio, it was found to have about 20% less capacitance and about 20% less coupled noise. At higher ratios used in some regions of the connector, the improvement is even more dramatic.

Optimized Backplane Cross-Section

Compliant-pin technology requires the use of a thick backplane to support the connectors. The 3.18-mm thickness is double the thickness typically used. Initially, this additional thickness was a cost detractor to the project. However, the additional thickness provides a performance enhancement to the product. Typically, four signal plane cross-sections have two orthogonal pairs of wiring planes outside of the power cores. In backplanes, wiring runs from slot to slot (horizontal); therefore, there is much less demand for vertical wiring. Although four wiring planes may contain enough wiring channels to support the product application, the orthogonal restrictions force the horizontal wiring layers to be dense and the vertical wiring layers to be sparsely populated. The density in the horizontal wiring layers makes it difficult to isolate sensitive signals and, also, may force wrong-way wiring on the vertical wiring layers to handle lines that do not fit on the horizontal wiring layers.

A new cross-section was developed using the additional backplane thickness. One orthogonal pair of wiring was moved to a central position in the backplane between two pairs of power planes containing only power or ground. This left one signal wiring plane on the outer two surfaces with no directional designation. The four wiring planes now better support the product wiring needs. In addition, if the separation between the horizontal and vertical planes in the center of the card is greater than the separation of those planes from their nearest power or ground plane, wrong-way wiring can also be done within this pair with much less penalty than in the usual case. In this cross-section, the total coupling between all lines in the adjacent layer and a test line is only about 30% of the near-neighbor coupling in the same layer. Thus, the orthogonal restriction on the central pair is not always necessary.

This cross-section has an additional advantage. Clocks and other sensitive signals are placed between power planes rather than on the surface, resulting in much better electrical characteristics.

Compliant-Pin Reliability

Demonstration of the reliability of compliant pins is a key achievement in this product introduction. Multiple compliant-pin designs are available on the market today. Some are proprietary; other designs are open. Extensive environmental testing was performed on the selected designs. Some tests simulated actual field conditions and others intentionally overstressed the contacts to induce failures so that compliant-pin failure modes could be understood. No contact failures could be attributed to the compliant pins, although backplane damage occurs when the defined backplane manufacturing specifications are not met. The final reliability assessment finds separable contacts to be the most vulnerable section of the connector system. The compliant-pin connections are as reliable as solder and have no measurable effect on system reliability. In fact, the elimination of all fluxing, soldering, and cleaning processes in the backplane assembly yields a higher quality product because of the process simplification.

Conclusion

Version 2 of the AS/400 system contains three backplane enhancements, two of which are directly related to the use of compliant pins. First, compliant-pin technology facilitates the manufacture of the double-sided backplane. This technology greatly increases the functions that can be placed in the processing unit and reduces the need for cabling to other expansion card enclosures. This is the core of the Version 2 system package that enables future system growth.

The second enhancement is the new backplane cross-section. This cross-section allows directional bias in backplane wiring to be easily accommodated. In addition, placing some signal layers between power planes rather than on the surface improves the electrical characteristics of the backplane.

The third enhancement to the package is the addition of the High Density Plus connector to critical locations in the backplane. This connector allows more usable signal pins than standard four-row connectors on the market because of the significant reduction of coupling and the reduced need for shielding in the pin field.

All of these functional enhancements have been made while enhancing the quality and reliability of this product through process simplification.

Manufacturing

The processes used to manufacture the AS/400 system were developed with the participation of manufacturing and test engineers from all locations. This ensures the ability to manufacture with consistent quality, using like processes at all manufacturing sites.

AS/400

MANUFACTURING

Early Manufacturing Involvement

Describes the roles that multidisciplined teams and early entry in the program cycle have in improving the quality and delivery schedules of product and process designs.

Thomas N. Paske, Stephen E. Wheeler, and Joseph T. Rommel

Introduction

IBM Rochester has worldwide development responsibility for the AS/400 system and advanced storage devices. It has manufacturing responsibility for those products in Rochester. The AS/400 products are also manufactured in Europe and Mexico and remarketed through IBM business partners in Brazil and Japan.

The processes used to manufacture the AS/400 products are developed with the participation of multidisciplined teams from all locations. This ensures the ability to manufacture worldwide with consistent quality, using like processes at all manufacturing plants.

Figure 118 depicts the overall manufacturing process flow of information, subassemblies, and completed systems.

The customer order drives the system assembly and test process. Subassemblies from the card and disk production lines are staged at the final assembly area in limited kanban quantities (**kanban** is a term used to describe a fixed quantity of inventory before and after a manufacturing process step). As the cards and files are consumed, pull signals are sent to the subassembly production areas for replenishment. This pull signal then drives the subassembly production.

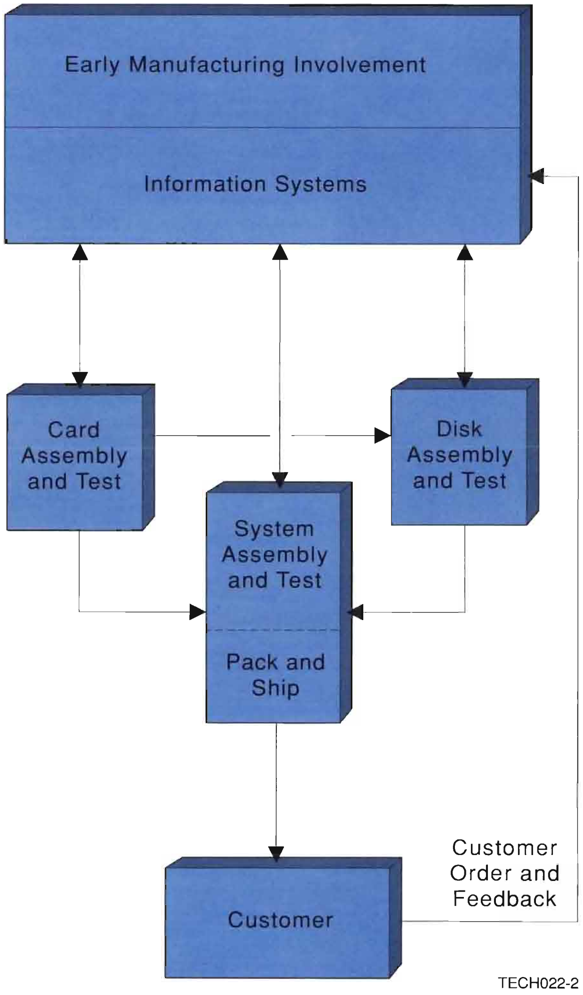


Figure 118. *Manufacturing Process Flow*

Rochester's manufacturing plant is capable of producing any configuration of any AS/400 model to a customized order on the same production line. There is no need to change setups between different system types (for example, 9404 System Unit to 9406 System Unit) or models of the same system type (for example, 9404 Model D10 to Model D20).

The manufacturing process emphasizes simplicity and provides expansion capability primarily through the use of manual operations. The process is driven by a local area network (LAN) controlled by an AS/400 system with attached IBM Personal System/2 (PS/2) computers. This low cost combination provides maximum flexibility for the scheduling, manufacturing, and tracking of all models of the AS/400 system as they move through the production facility.

To achieve these attributes, manufacturing places a concentrated effort on strategic design, early manufacturing involvement (EMI), continuous flow manufacturing, and computer-integrated control at an early stage in the product development cycle. Factors critical to the reduction in product and process complexity receive heavy emphasis:

- Design for modularity in product assembly. This includes the use of relatively large subassemblies that can be built and fully tested as major units before they reach the final assembly area. (A **subassembly** is a collection of parts that can be put together to form a stand-alone unit representing a major part of the final assembly or system. The card enclosures in the 9406 D models are examples of subassemblies.) Modularity yields easier assembly and part number reduction at the final build station.

- Reduction in part number count. Part numbers, in general, are reduced through the use of common parts in many designs; reduction of part numbers at the final assembly and test area is a direct result of both commonality and the modularity described above. Stocking, material movement, and tracking are simplified as a direct result of part number reduction.

- Reduction of part storage on the production line. **Pull logistics** is a method that replenishes stock in the production facility based on demand, not long-term plan) is the primary process used to reduce stock at suppliers, in transportation pipelines, and on the manufacturing floor. Parts inventory reduction increases flexibility, frees floor space, and provides efficiency in terms of cost and resource utilization.

Strategic Design and Early Manufacturing Involvement

The manufacturing team for each release of the AS/400 system is organized early in the development stage of the product design. The team mission is to assist in developing the design of the product family from the perspective of:

- Increasing overall ease of manufacture
- Improving the logistics (the procurement and transportation of material) flow of all materials in the product, from first-level suppliers to finished goods
- Reducing product manufacturing cost
- Enhancing shipped product quality in all design and manufacturing areas over previous releases

Design modularity is a critical success factor in achieving total manufacturing quality with high flexibility and low cost. Early involvement manufacturing and development teams build the product concept around pluggable subassemblies, or modules, testable at the unit level. This approach yields a major increase in the ability to customize the product set and yields a corresponding decrease in product complexity because a wide range of models can be produced using a set of common modules in different combinations.

The modular design concept also permits a reduction in part number count over products designed with more traditional approaches. Part commonality, part number reduction, uniformity of subassembly design, and modularity simplify part storage by reducing floor space needs. If the need of the customer set for specific models changes over time, the floor layout can remain relatively unchanged.

Modular design techniques also eliminate the need for setup between production of different models because all models use the same basic assembly approach. Because the part number count is reduced and the subassemblies are designed to be fully testable at a major unit level, supplier selection and material flow logistics are optimized and continuous flow manufacturing is realized.

The team builds integral aids for assembly into the design of the systems, modular subassemblies, and detail components. These aids include snap together parts, hand operable fasteners, and self-locating features that reduce assembly time and cost, and virtually eliminate the need for expensive assembly tooling at suppliers and at the product manufacturing plants worldwide. When the product complexity is reduced in this way, the skills necessary for manufacture decrease and the base of suppliers available for production of components increases in proportion.

Manufacturing's continuing emphasis on system design and the removal and prevention of defects early in the cycle creates continuous quality improvement. Early manufacturing involvement is approached with a multifunctional team philosophy and involves manufacturing engineers, process engineers, development engineers, procurement engineers, quality engineers, suppliers, personnel from non-US locations, and other groups as necessary. These teams apply concurrent engineering principles to the design of both the product hardware and the manufacturing processes, achieving a high degree of **optimization** (the point at which individual design and process elements mutually present the best combination of function, quality, cost, and delivery potential possible). Concurrent engineering allows the team to identify design and process defects and deficiencies that normally do not appear until the product is in production. Among the techniques are:

- Establishment of early manufacturing involvement teams during the product definition stage
- Simultaneous product and process concept evolution and design
- Continual multifunction reviews and information sharing
- Testing of designs and processes at many stages, beginning early in the cycle

As the team identifies areas needing change, plans are put in place to correct the deficiencies at their source. Because the team begins this activity early and continues it throughout the entire development cycle, corrections and changes are made when the product and process designs are still flexible. This process allows the optimal solution to a problem and creates minimal pressure on product schedules.

The team effort and simultaneous design and process evolution allow the optimization of both material and information movement in manufacturing. US and non-US suppliers and manufacturing functions are asked to provide their specific requirements of product and manufacturing process design regarding logistics optimization. The resulting input is built into the overall product strategy and implemented in the product and process designs. US and non-US suppliers are carefully selected to provide the highest possible degree of material and information movement efficiency. The early manufacturing involvement team evaluates suppliers using the following criteria:

- Full service manufacturing capability including the ability to design and test as well as manufacture

- Ability to produce fully tested major unit subassemblies in a wide range of commodities

- Ability to ship their product using a pull logistics system and to procure parts from their subcontractors using similar pull logistics techniques

- Capability in data collection and communications areas, particularly regarding quality systems and quality feedback

- Physical location and the complexity of the transportation network

Design modularity, the use of full service suppliers, and supplier input to product and process design are critical success factors in logistics optimization to meet the team's ultimate goal of continuous flow manufacturing.

The team approach enables true design and process matching for all aspects of the project to occur. Manufacturing involvement in the product design and development involvement in the process design create the feedback mechanisms so important in the mutual optimization of designs and processes. Manufacturing and development regularly schedule design reviews where product design change recommendations are communicated to development and process change recommendations are communicated to the manufacturing process groups. Emphasis is placed on the transfer of information, ideas, and requests from manufacturing, suppliers, and development to all those concerned in product and process design. Every part, assembly, and subassembly is examined for design functionality, manufacturability, delivery lead time, tooling requirements, shipping efficiency and expense, and quality considerations. The methodology of design and process matching:

- Establishes product and process designs with quality built in at the definition stage and continually verified as the design point evolves

- Provides protection of manufacturing investment by reuse of existing product and process design points and physical assets (change is, however, applied where needed to meet specific objectives and to protect our strategic future)

- Keeps customization potential high through modularity and reuse of subassemblies and customer orderable features while promoting manufacturing efficiency through part number reduction and logistics optimization

- Eliminates surprises and, therefore, allows the team to manage change in product and process design instead of reacting to it

- Improves communications between manufacturing and development at the earliest point in the design cycle and brings good ideas to the surface early

Conclusion

Both product and process design are significantly enhanced by the careful application of early manufacturing involvement. Early defect removal processes substantially improve quality and allow the management of, rather than the reaction to, change. Design and process matching and the resulting design modularity increase flexibility on the plant floor, shorten logistics pipelines, and reduce assembly time and complexity thus yielding improved manufacturing cost and shorter cycle times.

Manufacturing the System

Describes the system assembly, test, software preload, and shipping processes.

Christopher Guibert and Gary L. Stowe

Introduction

Never before has IBM placed more emphasis on meeting the needs of the customer.

IBM Rochester's commitment to this objective is reflected in its market-driven manufacturing system.

A manufacturing system must have the flexibility to meet diverse customer requirements, short cycle times for quick response, and perfect quality in process and product.

This article discusses the components of the market-driven manufacturing system in the areas of:

- Early process development
- Assembly
- Test and software preload
- System shipment

Early Process Development

The early process development procedure uses the early manufacturing involvement (EMI) concepts to develop the process. The manufacturing team's objective is to design a process that satisfies customer requirements while maintaining the integrity of established targets. One key attribute in producing a process that supports this objective is the process evolution coinciding with the product evolution. This methodology exercises the process prior to producing the first customer shipment.

With the emphasis on meeting and exceeding objectives, the manufacturing team constructed a process development laboratory adjacent to the final production facility (see Figure 119). This enabled the various disciplines to test and exercise their deliverables (process, parts, workmanship, and orders) in a semiproduction mode. Through the use of benchmarking, internal and external customer and product requirements, and strategic direction, manufacturing established targets during the process design stage. With the targets and simulation process in place, the team (including process, test, procurement, distribution, production control, and order logistics) was able to analyze and correct problems resulting from the emergence of a new product into a production environment. Along with trying to understand the process, the team expanded its view to that of a customer. Extended testing on the early systems provided insight for test enhancements. Early factory and field installations exposed potential problems and provided the manufacturing team with insight into customer perceptions. The combination of all the quality activities produced a priceless collection of data for analysis and tracking using the closed-loop method. This method is a key

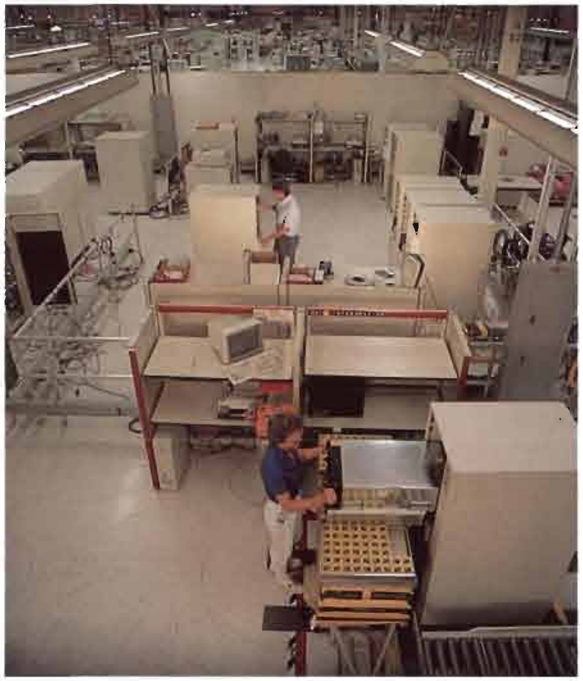


Figure 119. *Process Development Laboratory*

ingredient and uses resources from every participating AS/400 area of business (development, manufacturing, marketing, and field services) thus producing an integrated customer solution.

In addition to the quality benefits that the early process development exhibited, the AS/400 manufacturing team also decreased the manufacturing cycle time and improved process flexibility. The following are a few other contributions produced by the process development laboratory:

- Development of tools to improve operator intervention
- Design and simulation of process flows for maximum throughput
- Improved operator training and sense of work station ownership
- Increased production capacity capabilities (if required)
- Identification of design improvements to enhance manufacturability
- Improvement of the work station layouts and ergonomics
- Design processes to handle various models on the same production line

The combination of all these contributions produced a process that was implemented transparently into the production environment. This combination is a key attribute in the ability to manufacture a customer AS/400 order while maintaining cycle time integrity.

Assembly Process

The AS/400 assembly process is a low-cost production facility capable of producing any of the AS/400 models, to a customized order, with no setup time required between models. The process is manually oriented (actual assembly), providing optimum flexibility for change. The process provides controlled work direction from an AS/400 system for the operators. This creates a paperless environment from order input to shipment of the finished product.

The process begins by scrutinizing each customer in a knowledge-based order validation process. Upon passing this validation, the customer order is sent to the assembly station. Product and process errors are automatically flagged with the probable cause identified and displayed online to the analyzers.

Each operation is system controlled to minimize any potential human error. Quality is designed into the process. One of the methods used to promote the expected quality objectives is the in-process verification method. The control system manages the verification of the correct assembly by not allowing an operation to be completed until the correct part number is recorded and assembled. Quality focus teams analyze (online) potential defects also tracked by the control system. Bar coding of identified parts is a requirement in the assembly of the AS/400 system. To eliminate human error on vendor assemblies, the process uses electronic data interchange (EDI) to add the data automatically in the control system.

There are some applications where the AS/400 assembly process uses the concepts of image to eliminate potential process defects (see Figure 120). The floor-control system converts the customized instructions into a pictorial view for the operators. The image illustrates what parts are to be assembled (with description and color codes) and how they are to be assembled. The process is monitored on an ongoing basis against the quality targets established during the early process development stage.

Figure 120. *Verifying Location Using Image Technology*

A traditional manufacturing layout is designed around the product and concentrates mainly on product shipment. To meet the cycle time and flexibility objectives, the process flow—not the product—is the key design point. The continuous-flow-manufacturing concept is the basis for constructing the process flow. This concept entails a U-shaped line where parts come into one side of the U and the finished product exits on the other (see Figure 121).

With this type of flow, the problem of congestion and wasted product movement is eliminated (resulting in reduced cycle time).

Two other continuous-flow-manufacturing principles in the assembly process are operator ownership and building to a controlled kanban square. (**Kanban** is a term used to describe a fixed quantity of inventory before and after a manufacturing process step.) Operator ownership implies that one operator follows the product all the way through a defined set of operations. This greatly reduces the cycle time and work in process along with increasing the quality levels due to product ownership from start to finish. In addition, kanbans do not allow operators to start more work unless they need a specific part or assembly. This is programmed into the floor-control system to eliminate human judgement of what is needed. Using concepts that influence cycle time reduction and increase flexibility reflects a decrease in space, inventory and order delivery to the customer.

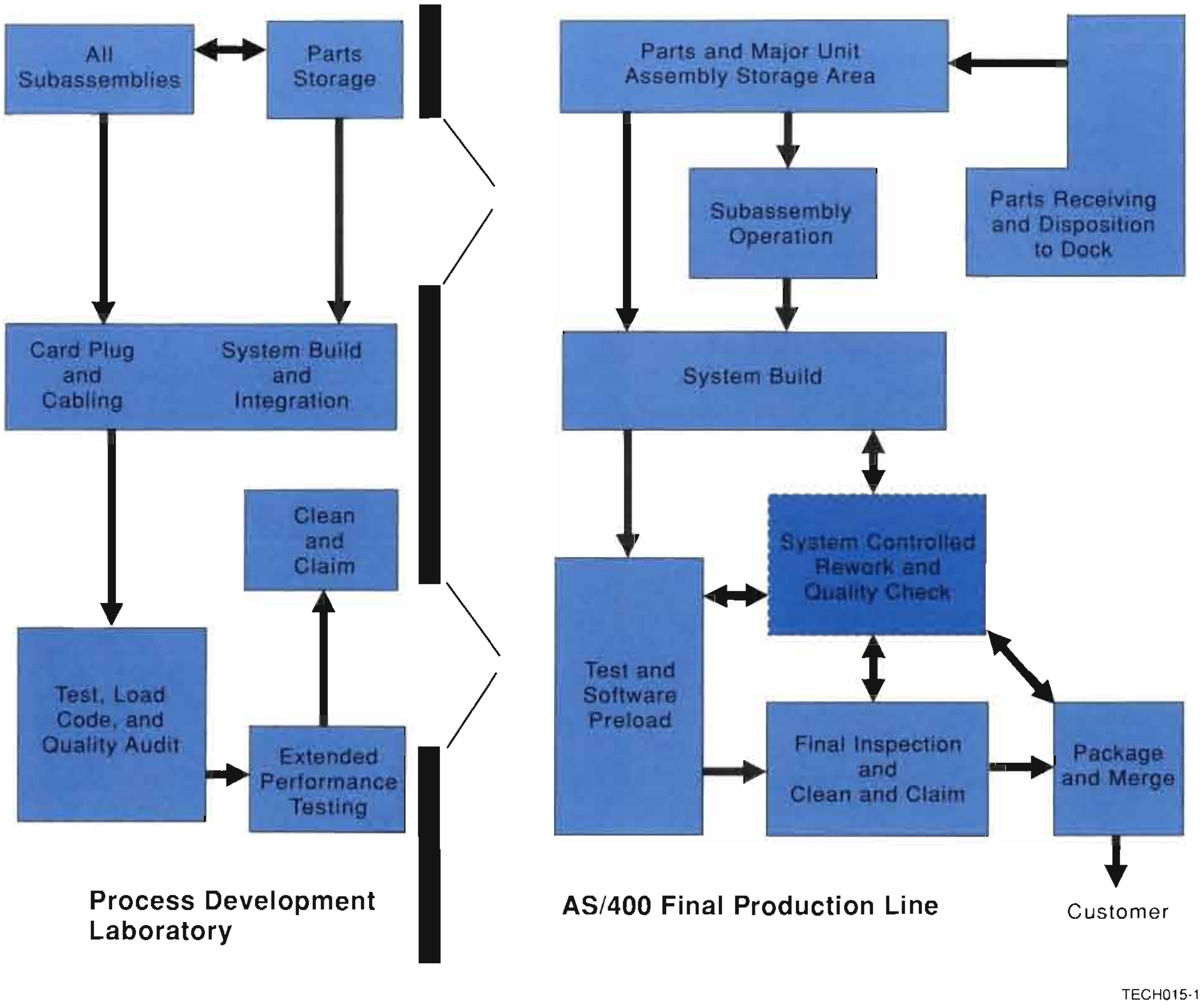

All Subassemblies
Parts Storage
Card Plug and Cabling
System Build and Integration
Clean and Claim
Test, Load Code, and Quality Audit
Extended Performance Testing
Process Development Laboratory
Parts and Major Unit Assembly Storage Area
Subassembly Operation
Parts Receiving and Disposition to Dock
System Build
System Controlled Rework and Quality Check
Test and Software Preload
Final Inspection and Clean and Claim
Package and Merge
Customer
AS/400 Final Production Line
TECH015-1

Figure 121. *Systems Assembly and Test*

System Test and Software Preload

While the product mechanical design has become somewhat standardized, testing the system and preloading software have become increasingly diverse. The number of hardware and software solutions are constantly increasing and becoming more customized. At the same time, the highest quality must be maintained.

To meet the challenges of increased solutions, the AS/400 manufacturing process is built around two key principles:

- Manufacturing to meet customer solutions
- Integrating quality-checking tools into the process

Manufacturing Process Designed for Solutions

The AS/400 test and software preload stations are designed around the customer requirements with simplicity as the foundation to maximum system flexibility.

Mechanical automation, typical of high volume and repetitive manufacturing, is minimized. Work stations, such as test cells, software preload stations, rework, inspection, and failure analysis, are integrated into one physical area. This integration minimizes product movement between work stations and eliminates the need for costly material-handling systems. The low-end entry AS/400 systems are assembled and tested on manufacturing carts. The manufacturing carts provide a flexible and simple means of transporting products throughout the process.

In addition to material-handling flexibility, the AS/400 test and preload stations are compatible with all products. IBM Personal Computers provide a flexible interface to the host floor-control system. The personal computers initiate test programs, relay information from and to the host system, and provide process and quality control.

The simple layout of the test and software preload stations allows operators to access the AS/400 system under test (see Figure 122). It is this access that enables custom and unique operations to be performed on the product as part of the normal manufacturing process. Rework actions are also simplified by allowing rework operators to observe the system under test and rework it at that time, without product movement.

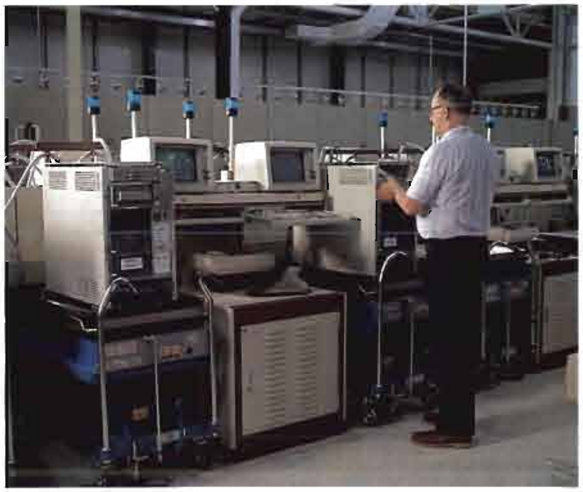


Figure 122. *Test and Software Preload Station*

The AS/400 software preload process is also designed to meet customer solutions. In the past, software was sent to customers on diskettes. Now, the AS/400 manufacturing system preloads the system software before shipment. A unique software preload process loads and distributes all the customer software evenly across the disks. This software preload process improves installation and setup for AS/400 customers.

In many cases, customers request unique versions of software or want to preload their own software. These special requests are accommodated easily in the AS/400 manufacturing system. In these cases, a customer's technical representatives and IBM manufacturing engineers meet to define the software preload requirements. Upon definition, part numbers for that particular customer are incorporated into the AS/400 bill of material structure. Thus, when that customer places an order, the part number automatically appears on the order and the appropriate software is preloaded.

Manufacturing Process Designed for Quality

Quality inspection is no longer a verification step at the end of the process. Rather, it is integrated into and throughout the process. It is the goal of this process to attain Six-Sigma (3.4 defects per one million opportunities) quality by improving the process a factor of ten in incremental steps. This goal is achieved by incorporating quality-checking tools in the process.

Throughout the assembly and test process, the manufacturing floor-control system continually validates part numbers, serial numbers, and engineering-change levels to coincide with the customer order. Order validation ensures that customers get exactly what they want when they want it.

Each work station on the manufacturing line is an inspection station. A dynamic inspection program, run at the first step of each operation, randomly asks questions of the manufacturing operators. These questions are changed as necessary, and feedback is given to the operators immediately.

Intelligent rework programs greatly aid in the rework process. If a failure occurs, the rework programs freeze the failure, analyze it against the error history, and recommend repair actions to the rework operator. As the rework operator makes the corrective action, the rework program is updated to reflect the latest repair actions. Components that fail in system test are immediately taken to a failure verification station. Here, failures are analyzed, verified, and communicated to the appropriate design team. This timely response provides an efficient and effective rework process.

However, the people are at the heart of manufacturing process quality. With a common goal of ten times improvement, manufacturing teams of operators and manufacturing engineers study the process to find opportunities for improvement. Each error is identified, analyzed, and acted upon for improvement. Process improvement teams, headed by the manufacturing employees, have shown significant results in process quality.

System Shipment

After the system is assembled, tested, and preloaded, it is ready for shipment. However, the AS/400 shipping process goes beyond traditional shipping processes by providing a Total System Package.

Total System Package

Here, the customer is provided almost all components of the system order: printers, displays, modems, cables, publications, blank tapes, and printer paper. This Total System Package, in one consolidated shipment, provides customers with all the parts necessary to set up and use their AS/400 system the same day it is received.

Shipping Process

A Total System Package demands system flexibility. Using diverse manufacturing resources, the shipping process handles many unique customer requirements to customize the system shipments (see Figure 123).

Within the shipping process, certain features ensure the highest shipping quality. One key element is that the shipping department is located next to the manufacturing department. As such, communication is enhanced, and expensive material handling is minimized.

Figure 123. *Hydraulic Lift Used for Product Movement*

Another key element is that components in the Total System Package contain bar code identification. Bar coding is a standard requirement placed on all IBM suppliers. When the system is ready for packaging, an operator scans the bar code of each part against the content of the customer order. The use of bar coding eliminates the human error element and improves quality (see Figure 124 on page 223).

Figure 124. *Bar Code Equipment in Use in the Packaging Process*

Once the components have been assembled to match the customer order, they are packaged with highly tested packaging material. Early in the development cycle, system packaging is stressed until it passes rigid environmental, drop-and-shake tests.

While all components of the system are packaged, shipping labels are applied, and the components are consolidated for shipment. Total cycle time for the process from start to finish is less than 24 hours.

Conclusion

A manufacturing process designed for customer solutions must adapt to a rapidly changing marketplace with a high quality product. Early process development maximizes manufacturing quality. While simplicity produces flexibility, inherent quality-checking tools enable the AS/400 manufacturing process to achieve its Six-Sigma quality goals.

Manufacturing Card and System Tests

Describes how early involvement and enhanced test equipment and processes enable the manufacturing group to deliver high-quality products at a lower cost.

Gary L. Kearns and Keith R. Halphide

Introduction

The manufacturing test objectives for Version 2 of the AS/400 system are to implement enhancements to the Version 1 process that support continuous progress toward improvement objectives.

The breakthrough process techniques introduced with Version 1 of the AS/400 system continue as the working platform for Version 2 enhancements. These include early manufacturing involvement and delivering stable, manufacturable designs to production by the following activities:

- Early stress testing using guardband functional assessment at card and system levels during the product development cycle
- Addition of optical and in-circuit card test capability
- Continued improvement of functional card testing capability established for Version 1
- Guardband product audit of ongoing production
- Ongoing system performance monitor benchmarking

Card and system manufacturing engineers participate in the development stage of the product to help ensure that a stable design is delivered to manufacturing. Manufacturing engineers work with development to stress test logic cards during early engineering tests. Functional card test, introduced at Version 1, continues to be used to reduce the number of test steps and to enhance the effectiveness of the test.

Early Manufacturing Test during Product Development

Early in the product development cycle, manufacturing engineers from Rochester and from other AS/400 manufacturing sites worldwide work with development engineers to select test methods and architectural requirements for product cards and testers. Product cards and testers are designed concurrently to support shortened product development cycles. Prototypes of production functional testers test many of the development engineering cards. Manufacturing and development engineers have joined together on some subsystems to such a degree that the initial subsystem operation and card characterization is done on manufacturing tester prototypes located within the development laboratory.

Early stress tests are performed on logic card assemblies from early engineering prototypes through the final design. Manufacturing engineers work with suppliers and developers to correct deficiencies and define custom production stress profiles for each logic card. An extra margin of safety, or guardband, is verified in this testing, ensuring that later manufactured parts obtained from multiple worldwide sources will operate properly through the full system specification range. Data from early production provides a baseline for process control and initial knowledge for the system that aids in defect diagnosis at functional test.

Guardband testing demonstrates performance capability beyond normal system specifications. It involves subjecting hardware to extreme operating voltages, temperatures, and oscillator frequencies to determine the actual functional limits of a given design. By doing early guardband testing while development engineers are still heavily involved, key technical people are available to diagnose and solve potential problems found during the tests. The early detection of problems allows time to modify designs, change supplier choices, and improve manufacturing quality before volume production begins. This approach can eliminate the need to depend on additional specialized stress testing as a screen to fine tune or selectively assemble systems to meet operating requirements.

Card Test during Production

Card level test begins after placement and soldering of parts to the raw card. Figure 125 illustrates the addition of in-circuit and optical test capabilities and the consolidation of functional test equipment by migration from dedicated to multipurpose functional testers.

In-Circuit Test

Enhancement of short circuit and open circuit test equipment allows in-circuit testing on selected logic cards. In-circuit testing supplies power to the device under test (DUT) and tests selected parts using signals supplied and monitored by the tester electronics. In-circuit diagnostic resolution is high because of the location of probe points throughout the DUT. Figure 126 on page 226 shows a bed of spring-loaded probes in contact with various points on the DUT for an in-circuit test. These tests complement functional testing by providing direct parametric measurements and by providing coverage for functional test configuration-dependent defects.

Optical Test

The use of optics in the AS/400 system brings new challenging technology to the card test arena. Figure 127 on page 226 shows the optical card tester. The tester provides laser testing to meet strict safety standards, as well as high-speed performance testing of the optical subassembly. This testing takes place prior to integration into the book package for functional testing.

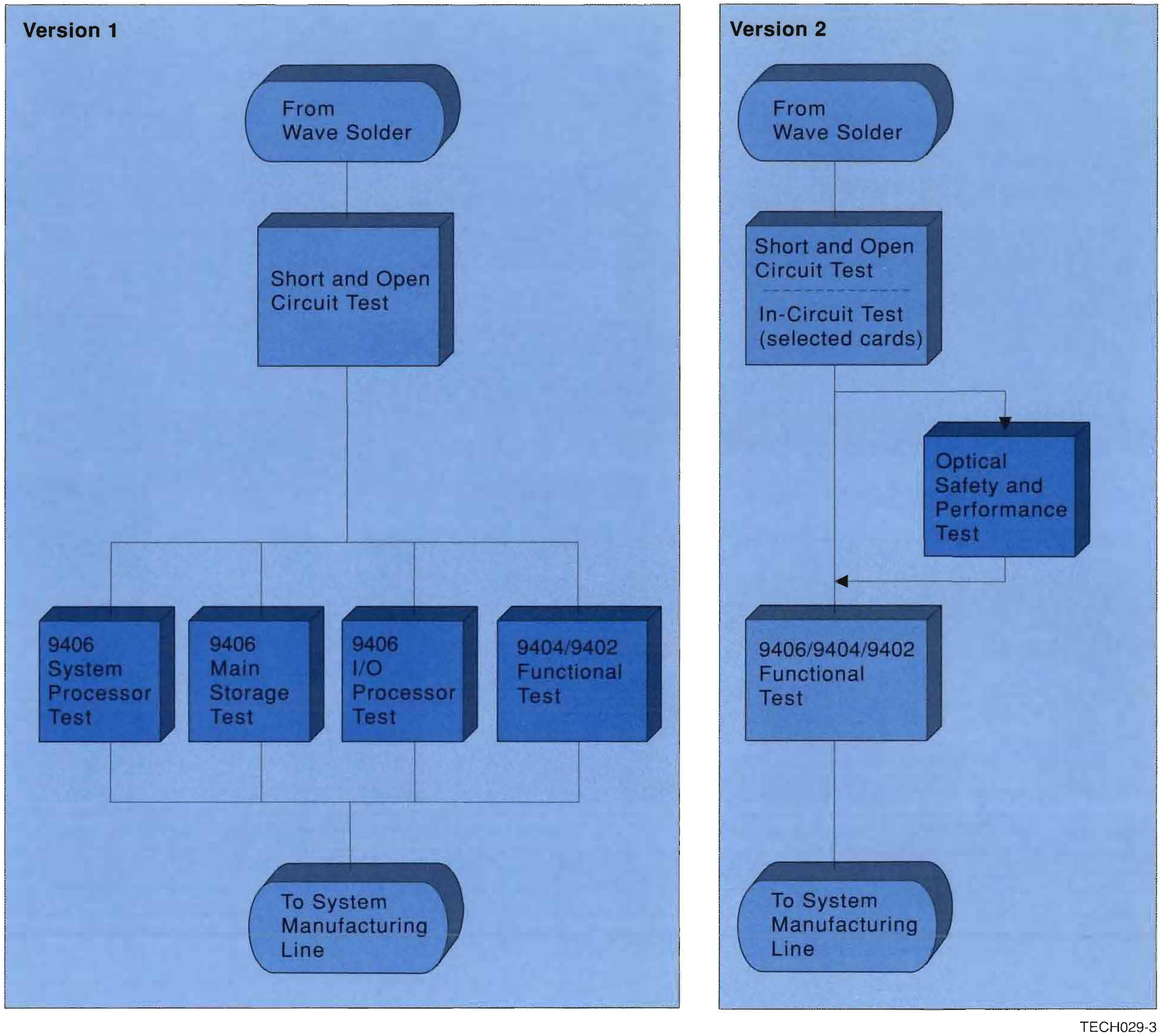

Version 1
From Wave Solder
Short and Open Circuit Test
9406 System Processor Test
9406 Main Storage Test
9406 I/O Processor Test
9404/9402 Functional Test
To System Manufacturing Line
Version 2
From Wave Solder
Short and Open Circuit Test
In-Circuit Test (selected cards)
Optical Safety and Performance Test
9406/9404/9402 Functional Test
To System Manufacturing Line
TECH029-3

Figure 125. *AS/400 Logic Card Test.* Each block represents a unique piece of test equipment. Test processes are shown within the block.

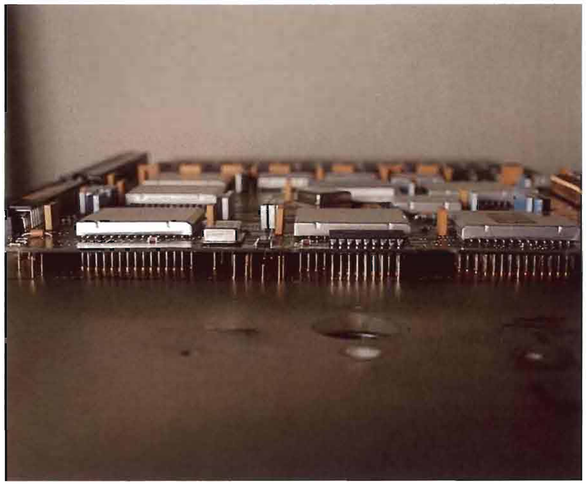


Figure 126. *Test Probes in Contact with a Logic Card*

Figure 127. *Optical Safety and Performance Tester*

Functional Test

The efficiency and effectiveness of functional test continue in Version 2. During a functional test, the card receives the same signals, timings, and instructions it would in an actual system running in a customer environment. Functional test utilizes the microprocessors on various AS/400 logic cards. The microprocessor tests itself, the logic contained on the card, and then the external buses and ports on the card. Wrap connectors, actual system devices, or tester logic cards attach to external buses and ports. The tester logic cards often make use of the same parts used in the system. Reuse of system parts provides real-life functional testing, as well as a savings in the design time and the cost of test equipment.

Main storage subsystem cards have no internal microprocessors to perform self-test. The tester logic card emulates the system main storage bus by providing and monitoring signals at system speeds. The tester logic card is capable of varying the timing parameters to perform further stress tests on the main storage bus as well as to record main storage addressing information of failures for supplier use in analysis and process improvement.

Version 2 brings boundary scan to selected AS/400 cards. This technology allows testing at full system speed, with the ability to stop the system clocks to observe and modify the operating states of latches controlling I/O signals. System diagnostics and functional card test both use boundary scan to control and monitor logic that is inaccessible to conventional circuit probes. Additionally, with the appropriate tester logic card, boundary scan provides extra information about the nature of a failure, which aids in card repair diagnostics and provides feedback to suppliers.

Figure 125 on page 225 illustrates that the four types of functional testers in Version 1 were reduced to a single functional tester in Version 2. Through migration of Version 1 card test software to new tester logic cards and the Personal System/2 (PS/2) computer, the Version 1 9404 and 9402 functional card tester has been upgraded to multipurpose functional testers at significant cost savings over building and supporting additional equipment. This consolidation and reuse of test equipment is key to meeting shortened product development cycle goals. Additionally, production flow is smoother because multipurpose testers accommodate variance in product mix better than dedicated testers. Figure 128 on page 227 shows an operator changing the functional tester to perform a test on a different model product.

Figure 128. *Multipurpose Functional Tester Setup*

System Test during Production

System test of the assembled AS/400 system is the final safeguard to prevent shipping a defective system to a customer. The integration process includes verifying that the assembled system configuration matches what the customer ordered. This includes all specified software loaded on the system at the latest change levels. Any customized code for special bid customers is also loaded. The variety of possible software packages is growing constantly.

Final test for Version 2 AS/400 systems includes several system performance assessment methods that provide an indication of system performance capability. System performance limits, determined jointly by development and manufacturing engineers during product development testing, are used as a benchmark for comparing current production system capability.

Guardbanding

Selected final test cells are configured to perform voltage-margin guardband testing as part of the production final test process. The test initially operates the system at previously established benchmark voltage levels (for example, ±10%). If it passes, the test is complete. If it fails, one of several lower levels (±7% or ±5%) is attempted to determine where the operation transition point occurs.

There are two benefits from this methodology. First, specific exception systems can receive extra test and analysis to improve their performance capability. Repairs are performed to bring the performance up to benchmark standards, which reduces the likelihood of a customer experiencing an intermittent problem with the system. Second, analysis of unique situations that are detected leads to improvements in design and manufacturing processes that benefit future systems.

Guardband failures are closely analyzed by the cross-functional failure analysis teams through root cause analysis to determine causes for marginal failure situations. This information is then used to further evaluate the design and process capability as appropriate. Further validation of the guardband assessment process is performed by correlating subsequent actual field performance against the guardband margin capability exhibited during system production test.

Performance Benchmarking

Running performance monitor programs provides another benchmarking technique. These programs are a series of different customer data processing applications that benchmark processing time per system. The applications include commercial end-of-day closing transactions, RPG compilation jobs, and database query transactions. Performance benchmarking provides a method of quantifying the degree of similarity in performance capability between systems.

The various performance analysis applications described expand the traditional system test pass or fail assessment. More specific information is available for making system shipment decisions.

Partnerships

Rochester system test and card test share a common database, simplifying the correlation of information. Rapid card retest is provided within Rochester to support failure analysis. Improvement teams cross organizational boundaries to analyze failures and work with suppliers to prevent defects at their source.

Data collected from the system and card test and repair processes are analyzed for clues to potential process improvements. Suppliers and designers receive feedback from this database, along with reports from failure analysis teams, to assist in isolation and correction of problems.

Conclusion

Manufacturing card and system test emphasize proactive quality for the following reasons:

- To avert potential problems by active participation early in the design cycle of the product.
- To aid suppliers in isolating problems and correcting the source of the defect instead of simply separating the good from the bad.
- To satisfy customers by enhancing processes and communications, while delivering a competitive, quality product.

AS/400 logic cards receive functional test to assure the delivery of the highest quality assemblies. Teamwork between development and manufacturing engineers early in the product cycle assures the product is designed for manufacturability. Functional stress test of early production accelerates the detection of defects and the improvement of designs, processes, and parts before high-volume production begins.

System testing of custom configurations to each customer order assures customers receive the complete hardware and software solution ordered. Various performance benchmark or guardband audits provide assurance that expected system functional capabilities are delivered.

Supplier and customer partnerships with constant communications and feedback are essential to achieving customer satisfaction and quality improvement.

Data Management System of the Circuit Package Production Center

Describes the quality data management system, which gives complete defect traceability on AS/400 circuit cards.

Richard A. Saltness and Steven A. Horejsi

Introduction

The information systems in manufacturing are changing to match the flexibility needed in today's manufacturing environment. This article discusses how they are changing and an instance of that change in the data management system of the circuit package production center (CPPC). CPPC assembles and tests many of the circuit cards that go into the AS/400 line of products.

The information systems in Rochester manufacturing are the product of the 30 years that Rochester has been a manufacturing site. Today, floor control consists of a large variety of systems ranging from old to state of the art.

Clearly, as cycle time is further reduced on the manufacturing floor to the 1-day timeframe, nonintegrated batch systems become a bottleneck to the manufacturing floor. The way CPPC stores and delivers data must be flexible and responsive to a rapidly changing environment.

CPPC in Rochester manufacturing has made a sizable investment to move to the flexible, responsive information system needed to deliver the product to the customer quickly and with high quality.

CPPC Data Management

The card manufacturing process is a complex series of operations spanning a wide variety of specialties. The card process technologies supported by CPPC span the entire spectrum of card production, from custom raw card fabrication and test through the assembly, solder, and test of pin-through-hole (PTH) and surface-mount-technology (SMT) card assemblies. The specialized fields practiced within CPPC include chemical, electrical, and mechanical engineering, metallurgy, and process control to mention a few. The cards produced in Rochester are destined for both US and non-US locations for use in a wide variety of products. It is vital to Rochester's continued profitability that the quality, reliability, and availability of these card assemblies be maintained.

Former Data Collection System

With this broad mix of technologies, it is possible for specific process parameters to go beyond specified control limits. Inspections, test operations, and other safeguards are designed into the process flow to avoid the delivery of a defective product to customers. In the former data collection system, process and defect data was recorded on a so-called travel sheet, which indeed traveled through the process with a job lot of cards. While the previous method had the advantage of minimal cost, the penalties were many. First, the operator wrote the information on the sheet, which is both time-consuming and error prone. Second, a nonmanufacturing person entered the data into a batch system for overnight processing into the CPPC database; this introduced additional errors and labor costs. Also, the travel sheets were subject to destruction or misplacement, in which case, all data for a particular job lot was lost.

But perhaps the largest disadvantage of this batch environment was the inherent delay in the availability of information. Because the travel sheets followed the job, this data could not be entered into the database until the job was complete; the database did not contain any information about jobs in process. At the time, the average job cycle time was in the range of 3 to 4 days, meaning that processes gone awry went undetected for days because the information that could expose them was traveling about the manufacturing floor on a sheet of paper. The costs of these delays were in the form of wasted material, added labor, and lost time.

Finally, the collected information was stored in a database in nonstandardized format. This method made access to information almost as difficult as its collection. The people that needed the information to improve the processes had difficulty getting it.

Strategy Definition

A task force was formed to discuss and gather requirements for data management of the CPPC area. This task force exposed key areas of needed improvement and made immediate recommendations defining priorities in the area of data management.

An additional challenge was to provide quick and convenient access to the database. A large number of tools, reports, and programs were already in existence. These ran in a wide variety of environments and were written for the database. It was not the task force's intention to force users to learn new tools; any new process would have to operate effectively with as many existing tools as possible.

Implementation

After the task force generated the requirements, CPPC formed a team of key technical individuals to help gather requirements before code design. This team was the data collection steering committee.

Representation in this committee from over 17 different CPPC areas included test engineering, process engineering, test manufacturing, process manufacturing, CPPC business office, maintenance, quality, management, programming, vendor card procurement, and the early manufacturing involvement (EMI) team. These key people, who owned the key processes identified by the task force, as well as management provided the commitment needed. The task force assigned each action item to an owner. The owner was responsible to gather and define detailed requirements to resolve that action item.

It was the committee's responsibility to approve high-level designs and to ensure common procedural solutions as each of the action items was solved. The committee also reviewed and approved the system architecture required for future expansion.

This review process helped ensure that a delivered product fulfilled expectations of requirements gatherers, and provided a forum for feedback as requirements were implemented.

The solution was an integrated data management system that cooperatively collected, verified, and managed the data in a distributed relational database. This system enabled immediate data access, with total system latency averaging less than 15 minutes. Using this improved system, process problems were detected and corrected in minutes rather than days. Also, the accuracy of the data was significantly improved because only choices that made sense for the operation were presented to the user for entry, eliminating the opportunity for entry of nonsensical data.

Current Data Management System

Because a major goal of the data management system is to enhance process control and defect prevention, the timely and convenient availability of information is of primary concern. Also, because business decisions are based on this data, the accuracy and level of detail are of great importance.

An important paradigm in the process of system specification is considering CPPC an external IBM customer; that is, rather than write all of the necessary solutions, CPPC would attempt to integrate off-the-shelf IBM licensed programs as well as business partner solutions. This means that large pieces of the system would be maintained by IBM and its business partners rather than CPPC. The advantages of using off-the-shelf software are significant; fixing bugs and many new enhancements would be automatic. This means that programmers could concentrate their efforts on the customized pieces of the data management system yielding significant leverage in development effort.

The system itself consists of three major parts. First, the work stations run OS/2 Extended Edition in a token-ring network. These stations serve to integrate the numerous systems and applications present on a modern manufacturing floor, presenting the operator with the preferred single system view.

The second major part of the system is a small group of Personal System/2 (PS/2) systems that provide an area controller function on the token-ring network. The area controller distributes network services and shared files as well as provides a local relational database, which was implemented in the Operating System/2 (OS/2) database manager. This provides for continuous system availability, even when the host database system is down or stopped for maintenance.

The third major part of the system is the host database, which is implemented in a DATABASE 2* (DB2*) relational database on a Multiple Virtual Storage (MVS) platform. The host provides a powerful data storage and access environment and supports a variety of access tools, such as Statistical Analysis System (SAS) and Query Management Facility (QMF*).

Figure 129 shows the system architecture of the CPPC data management system.

Basic Operation

Each operator claims the work performed on each job of cards; this is called a process claim. If an operator finds a defect of any kind, the operator enters the defect claim into the system through a graphical interface that serializes the transaction and updates the DB2 data management system to minimize errors in the data.

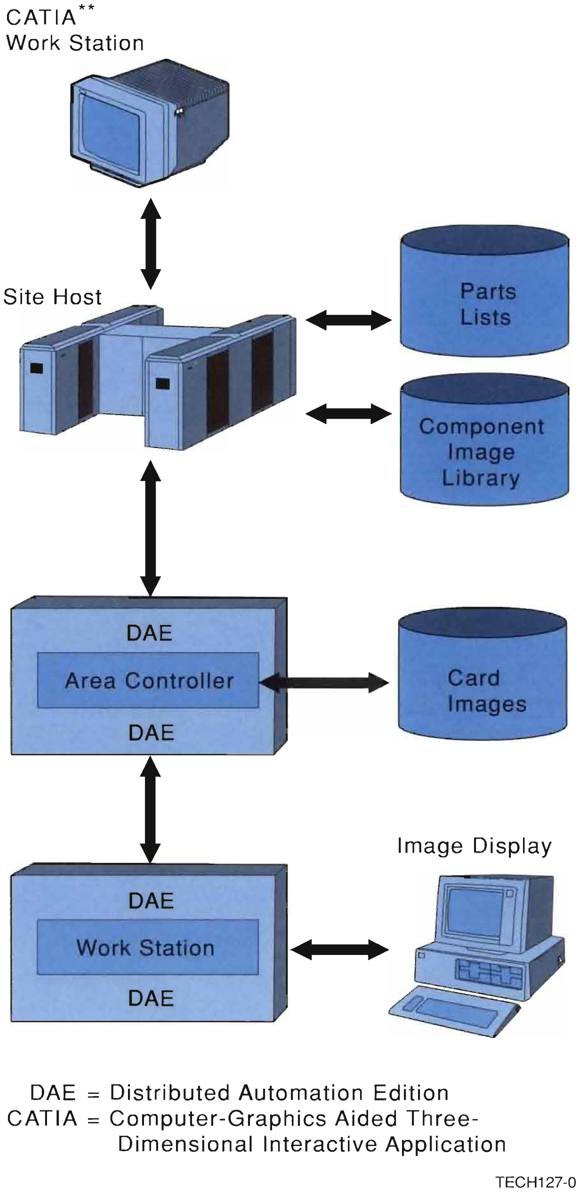
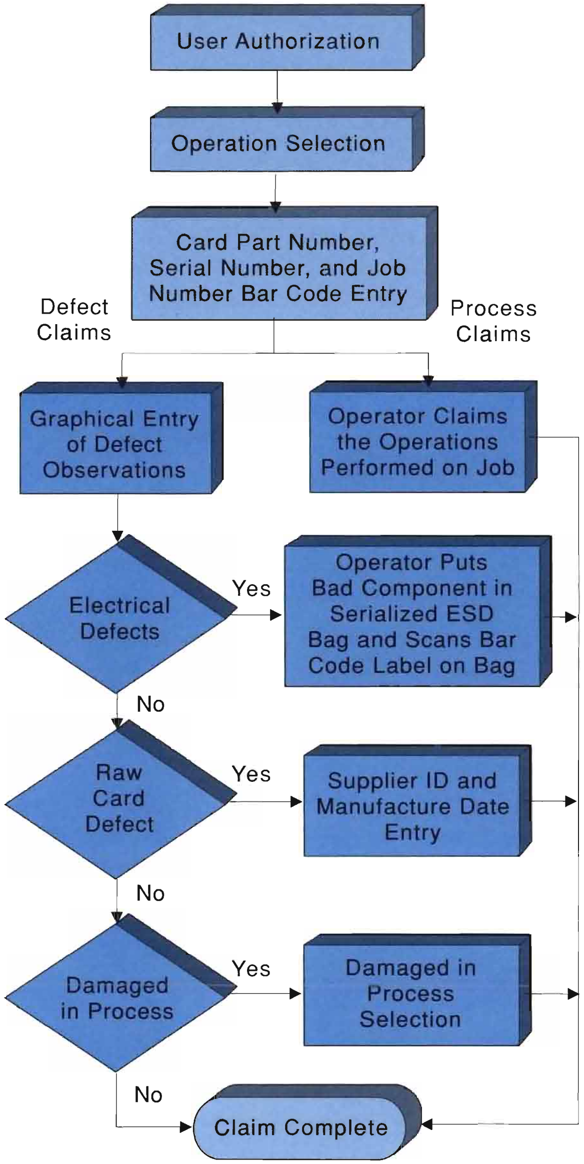


Figure 129. *System Architecture*

Figure 130 on page 233 shows the steps necessary for an operator to enter a claim in the system.

The graphical interface consists of a Presentation Manager image of the circuit card with its electrical components on it. Each electrical component is a unique bit map, and the image of the assembled circuit card is a collection of all of the component bit maps overlaid to become one assembled image. The component bit maps are stored in a component image library. When the development laboratory releases a new card to manufacturing, a series of programs analyzes the data and, based on component types, size, and location information, automatically generates the image. The image is then copied to our area controller to allow for better performance in delivering this data to the operator.


User Authorization
Operation Selection
Card Part Number, Serial Number, and Job Number Bar Code Entry
Defect Claims
Process Claims
Graphical Entry of Defect Observations
Operator Claims the Operations Performed on Job
Electrical Defects
Yes
Operator Puts Bad Component in Serialized ESD Bag and Scans Bar Code Label on Bag
No
Raw Card Defect
Yes
Supplier ID and Manufacture Date Entry
No
Damaged in Process
Yes
Damaged in Process Selection
No
Claim Complete

ESD = Electrostatic Discharge
ID = Identification

TECH019-4

Figure 130. *Operator Data Entry*

Figure 131 shows the graphical image of a circuit card used to make defect claims.

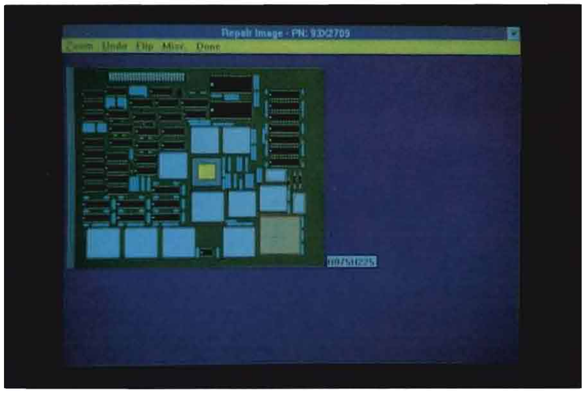


Figure 131. *Graphical Card Image*

Traceability

Each defect type is recorded against the operation that most likely caused it. Defect reports run four times per day and are delivered to the two lead floor technicians in the manufacturing area. These reports provide notification of process, maintenance, and operator education problems that cause multiple defects.

To decrease electrical component failures, CPPC now has traceability of failing components to the failure it caused on the assembled cards. This information is essential to failure analysis engineers and component suppliers.

Conclusion

The CPPC data management system is just one example of the ultimate goal of a totally integrated manufacturing environment across multiple manufacturing lines in Rochester. Similar development is under way in many hard-file manufacturing lines and the systems manufacturing line.

CPPC programmers consistently followed hardware architectures, communications protocols, and database practices, which makes linking information across manufacturing lines a straightforward process.

Authors

Paul V. Allen
Mr. Allen is a staff programmer in AS/400 programming development. He is the team leader for the user interface manager (UIM) component of the OS/400 operating system. Past assignments include programming on the UIM and the System/36 SSP. Mr. Allen has one US patent pending and two inventions published. Prior to joining IBM, Rochester, MN, in 1984, he received a BS degree in Computer Science from Western Washington University, Bellingham, WA, and worked at Dealer Information Systems, Inc.

James E. Bahr
Mr. Bahr is a senior planner responsible for system price/performance on the AS/400 system. Prior to this assignment he was a performance analyst in the Rochester Programming Laboratory, focusing on multiprocessor performance. Past assignments include technical and managerial positions in the Rochester Engineering Laboratory on processor development. Mr. Bahr is credited with one patent and has published two papers on multiprocessor performance. He joined IBM in 1973 after receiving a BSEE degree from the University of Minnesota.

Mark L. Bauman
Mr. Bauman is a staff programmer in AS/400 communications development working in the heterogeneous connectivity area. He is a member of the TCP/IP programming team. Mr. Bauman previously held various programming assignments on the System/38 and tools development group. He graduated from Mankato State University of Minnesota with a BS degree in Computer Science and Business with a minor in Mathematics.

Jonathan P. Beierle
Mr. Beierle is an advisory programmer in AS/400 communications development. Before joining the communications area in 1990, he worked on research projects in the advanced technology area involving programming development environments, programming languages, and communications. Prior to this, he was responsible for System/38 control language compiler development. He joined IBM in 1980 after receiving a BS in Computer Science and Statistics from the University of Wisconsin-LaCrosse.

Neil C. Berglund
Mr. Berglund is a senior engineer in power systems. In his most recent assignment he was responsible for the design and definition of the AS/400 intelligent power control network. Mr. Berglund joined IBM in 1965 and holds a Bachelor of Electrical Engineering degree from the University of Minnesota. He is credited with 12 patents and numerous disclosures for work on processor and I/O development in System/3, System/38, and the AS/400 system.

Bruce L. Beukema
Mr. Beukema is an advisory engineer in AS/400 processor development working on processor I/O bus development. Past assignments include various I/O related development in System/34, System/36, and AS/400 engineering. He holds two US patents and has 11 inventions published. He joined IBM, Rochester, MN, in 1977 after receiving his BSEE degree from South Dakota School of Mines and Technology, Rapid City, SD.

Timothy R. Block
Mr. Block is an advisory engineer in interconnect technology working on fiber optic data link development. Past assignments include component reliability testing and optical storage development. He holds five US patents and has nine inventions published. Mr. Block joined IBM, Rochester, MN, in 1974 after receiving a BS in Physics and Mathematics from Carroll College, Waukesha, WI, and an MSEE from Northwestern University, Evanston, IL.

John M. Borkenhagen
Mr. Borkenhagen is a staff engineer in automation technology. He is a team leader working on design automation of timing analysis and verification. Before joining automation technology in 1989, he had a variety of assignments in AS/400 processor design. He has filed two patents and has published eight disclosures. Mr. Borkenhagen joined IBM, Rochester, MN, in 1984 after he received

BSEE and MSEE degrees from the University of Wisconsin, Madison, WI.

John M. Broich
Mr. Broich is a staff programmer in the SQL/400 development group. He is currently working on SQL/400 run-time development. Mr. Broich joined IBM, Rochester, MN, in 1989 after working as a programmer for UNISYS Corporation. He has a Computer Science and Business BS degree from Mankato State University, Mankato, MN.

Laura J. Bruner
Ms. Bruner is a senior associate programmer in the application technology group. She joined IBM in 1988 after working as a programmer/analyst for State Farm Insurance. She provides education, application development support, and technical support for the CallPath/400 product. She previously worked with the Telephony Application Services product and the IBM 9270 Voice Response Unit. She received a BS degree from the University of Wisconsin at Stevens Point in 1984.

Ann M. Bukowski
Ms. Bukowski is currently an associate information developer in the application enabling interface group. She previously wrote PC Support/400 documentation. She joined IBM in Rochester, MN, in 1988. Ms. Bukowski has a BS degree in English and an MA degree in Business and Technical Communications from Iowa State University, Ames, IA.

David G. Carlson
Mr. Carlson is a staff programmer in communications protocol development whose current responsibilities include AS/400 X.31 and ISDN communications development. Past IBM assignments include AS/400 X.25 communications and I/O processor initialization development. Mr. Carlson joined IBM, Rochester, MN, in 1984 after receiving his BS in Computer Science from Southern Illinois University, Carbondale, IL.

David A. Christenson
Mr. Christenson is a staff programmer in AS/400 communications development. His previous assignments include design and code development for System/36 APPC, System/36 APPN, System/36 MLU, PC Support/36, AS/400 APPN, and AS/400 SNA. He joined IBM, Rochester, MN, in 1980 after receiving his BS degree in Computer Science from Moorhead State University, Moorhead, MN.

Jeffrey A. Collett
Dr. Collett is an advisory engineer in packaging technology responsible for the definition of packaging technology for future systems. Assignments since joining IBM Rochester in 1984 have been in connector and solder interconnect reliability evaluation. He joined IBM at the Thomas J. Watson Research Center in Yorktown Heights, NY, in 1983. Dr. Collett received a BA degree in Physics and Mathematics from St. Olaf College and holds AM and PhD degrees in Physics from Harvard University.

Kathryn D. Cook
Ms. Cook is a senior associate programmer in the database area of AS/400 development in Rochester, MN. She is currently working on the remote SQL function, the shared folders function, and the transfer function of PC Support/400. She joined IBM in 1986 after graduating from Case Western Reserve University, Cleveland, OH, with a BS degree in Computer Engineering.

Kenneth A. Cook
Mr. Cook is a senior engineer on the communications programming development team working in the area of heterogeneous connectivity. He was responsible for the software structure design and implementation strategy for the integration of the OSI product on the AS/400 system. His latest responsibilities included design responsibilities in TCP/IP communications, ISDN, telephony, OSI, and X.400. He joined IBM in Endicott, NY, in 1957. He managed the development of the remote communications subsystem for the System/38 and has held numerous engineering and programming responsibilities for the Rochester systems. He received the IBM invention award for two US patents and seven published disclosures.

Mark A. Cossack
Mr. Cossack is a senior associate programmer on the APPN development team. He joined IBM in 1989 after working on OSI and SNA networking products development at Control Data Corporation. Since joining IBM, he has worked on AS/400 APPN development in addition to other AS/400 communications projects. Mr. Cossack holds one US patent and has a technical disclosure published. He received a BA in Quantitative Methods and Computer Science in 1984 from the University of St. Thomas, St. Paul, MN.

Dennis T. Cox
Mr. Cox is a senior engineer working in circuit technology on various ASIC design systems.

Past IBM assignments include SRAM designs, programmable logic array (PLA) design systems, growable array macros, and CMOS and BICMOS design systems. Mr. Cox joined IBM, Kingston, NY, in 1970 after receiving his BSEE degree from the University of Wisconsin, Madison, WI. He received his MSEE from Syracuse University, Syracuse, NY, in 1975. Mr. Cox transferred to Rochester, MN, in 1976.

Thomas R. Crowley
Mr. Crowley is an advisory programmer responsible for the maintenance of the storage management portion of the system licensed internal code. Past IBM assignments include the development of various System/38 diagnostic programs (including the hardware system test and the pack utility maintenance program), and design and development responsibilities for the AS/400 disk maintenance and recovery procedures, the AS/400 save storage function, and the AS/400 disk failure status notification function. Mr. Crowley has numerous technical disclosures. Mr. Crowley joined IBM, Rochester, MN, in 1979. He received a BS degree from the University of Wisconsin at LaCrosse.

Robert L. Dick
Mr. Dick is an advisory programmer in the advanced systems I/O and communications group. He has been involved with various communications implementations on both the System/38 and AS/400 systems including both SNA and TCP/IP protocols. Mr. Dick joined IBM in 1981 after he received his BS in Computer Science from California Polytechnic State University, San Luis Obispo, CA.

Steven M. Douskey
Mr. Douskey is a staff engineer in advanced systems engineering. He is the lead designer for the AS/400 system processor self-test. Past IBM assignments include AS/400 system processor problem analysis and resolution (PAR), AS/400 processor bus adapter interface hardware, and System/38 processor channel interface hardware. He holds two US patents and has published 12 disclosures. He joined the advanced systems development group in IBM, Rochester, MN, in 1982 after receiving a BSEE degree from the University of Nebraska, Lincoln, NE.

Dana M. Duffield
Mr. Duffield is an advisory programmer in OS/400 programming development. Mr. Duffield joined IBM in 1982 after receiving a BS degree in Computer Science from the University of Illinois, Champain/Urbana, IL. Mr. Duffield is currently working on graphical user interface enabling for the OS/400 operating system. Past IBM assignments include development of nonprogrammable work stations and various development and lead design responsibilities on AS/400 PC Support. His work experiences prior to joining IBM include Bell Laboratories in Naperville, IL.

Randy L. Egan
Mr. Egan is an advisory programmer in the SQL/400 development group. He is the team leader for the SQL run-time support team. Mr. Egan joined IBM, Rochester, MN, in 1984 after working as a programmer for Deere & Co. He has a BS in Comprehensive Mathematics and Computer Science from the University of Wisconsin at Platteville.

Karen S. Eikenhorst
Ms. Eikenhorst is an advisory programmer in OS/400 programming development. Past IBM assignments include testing and programming for the System/34, programming for the System/36 control storage processor, user interface design for the System/36 and the AS/400 system, and development manager. Ms. Eikenhorst is currently working on graphical user interface enabling for the OS/400 operating system. She joined IBM in 1977 after receiving a BS degree from Iowa State University, Ames, IA.

Richard G. Eikill
Mr. Eikill is the manager of the system simulation group in advanced systems management. Past IBM assignments include the development of processors and main storage for both the System/36 and the AS/400 system. He holds seven US patents and 11 invention disclosures. Mr. Eikill joined IBM in 1984 after receiving a BSEE from the University of North Dakota.

John R. Elliott
Mr. Elliott is a staff engineer in design systems responsible for test generation tools and support. Past IBM assignments include logic designer on 3480 Tape Subsystem, test equipment designer for functional self-test tester, engineering design system center of competence, department manager of card test engineering, and technician. Mr. Elliott joined IBM in 1966 at Boulder, CO. He graduated from the University of Colorado at Boulder with a BSEE. He holds one patent and has published two articles in the IBM Technical Bulletin.

Earl W. Emerick
Mr. Emerick is an advisory programmer and manager of a group of AS/400 systems management and SystemView solutions. He joined IBM in 1977 after receiving his BS degree in Computer Science from the Indiana Institute of Technology, Ft. Wayne, IN. He held various assignments in software development on System/38, System/36, and the AS/400 system, where he also managed the development of AS/400 change management functions. His career has been focused on advancing availability, reliability, serviceability, and systems management technologies through software development, architecture, design, and management positions at many different levels of the system.

Dennis J. Frett
Mr. Frett is currently a senior associate programmer with the AS/400 APPN development team in Rochester, MN. Since joining IBM in 1987 he has also worked with AS/400 APPC development. Dennis received a BS degree in Computer Science from Iowa State University in 1987.

Janice R. Glowacki
Ms. Glowacki is a senior associate programmer working on the PC Support/400 and RUMBA/400** products. Prior to her current assignment, Ms. Glowacki was a developer working on the OS/2 communications manager. Ms. Glowacki joined IBM, Rochester, MN, in 1988. She earned both her BS and MS degrees in Computer Science at Florida International University, Miami, FL.

Christopher Guibert
Mr. Guibert is staff engineer responsible for AS/400 manufacturing process development and engineering. He joined IBM, Rochester, MN, in 1984 and has worked in automation and assembly engineering. He received a BS in Mechanical Engineering in 1984 from UCLA and an MS in Manufacturing Systems Engineering from Lehigh University in 1987.

Keith R. Halphide
Mr. Halphide is a staff engineer in the circuit package production center responsible for functional tester development and application verification. Previous assignments include stuck fault logic tester design, self-test programming, and disk burnish/test. He joined IBM Rochester in 1983 after graduating with a BSEE from the University of Washington, Seattle.

James J. Herring
Mr. Herring joined IBM, Rochester, MN, in 1981 from Bemidji State University where he graduated with a BS in Mathematics and Computer Science. His previous assignments include System/38 performance, implementing AS/400 token-ring LAN support, and leading the design and development of the software support of AS/400 ISDN. He is currently a development programmer and manager for software installation development on the AS/400 system.

Steven A. Horejsi
Mr. Horejsi is a senior associate engineer in the CPPC area. He is the database administrator for the CPPC data management system. Past IBM assignments include the development of logic card testers and card tests. He joined IBM, Rochester, MN, in 1978.

Merle E. Houdek
Mr. Houdek is currently a senior engineer engaged in performance analysis and modeling of processor and I/O hardware. He joined IBM in 1964 and holds a BSEE degree from Tri-State University in Angola, IN. He is credited with seven patents and numerous disclosures for work on processor and I/O development in the System/38 and AS/400 systems.

Randall S. Jensen
Mr. Jensen is a staff engineer in the main storage card development group. Before joining IBM, Rochester, MN, in 1988, he was a member of the technology applications group in Poughkeepsie, NY. Past IBM assignments include concurrent maintenance development for the System/370 channel cards and on-chip temperature sensing. He joined IBM in 1982 after receiving a BSEE degree from Michigan State University, East Lansing, MI.

Charles L. Johnson
Mr. Johnson is a senior engineer for circuit technology in AS/400 systems logic. In his current assignment he is responsible for setting the direction and development of the technical design methodology, analysis, and application of VLSI technologies used in future AS/400 processor products. Past assignments include analog and digital circuit design, chip logic design, future packaging technology concepts, system clocking methodology, and technology applications and delivery. He is credited with five US patents, 18 published inventions, 12 conference presentations, and four technical reports. He joined IBM in 1974

after he received a BSEE degree from the University of Minnesota, Minneapolis, MN.

Kenneth E. Johnson
Dr. Johnson is a senior engineer in recording media technology with interests in magnetic and physical properties of thin films. His career has been involved with rigid disk technology since its inception in Rochester. He plays a significant role in all of IBM Rochester's disk products with particular emphasis on the thin-film disk first shipped by IBM in 1988. He received a PhD in Physical Chemistry from the University of Minnesota in 1977. After a postdoctoral appointment at the University of Chicago, he joined IBM Rochester in 1988.

Sarah R. Jones
Ms. Jones is a senior associate programmer in the heterogeneous connectivity development organization. Since joining IBM in 1987 she has had various assignments involving AS/400 communications, and, most recently, design and development for user-defined communications. Ms. Jones received her BA degrees in Mathematics and Computer Science from Luther College, Decorah, IA.

Brian E. Jongekryg
Mr. Jongekryg is a staff programmer in communications protocol development whose current responsibilities include AS/400 SDLC and ISDN development. Past IBM assignments include System/36 SDLC and X.21 communications development. Mr. Jongekryg joined IBM, Rochester, MN, in 1984 after receiving his BS in Computer Science from Michigan State University, East Lansing, MI.

Kerry T. Kaliszewski
Mr. Kaliszewski is a senior associate engineer in processor self-test. His previous assignments include writing diagnostic licensed internal code for the AS/400 system as well as for IBM PC video test equipment. He joined IBM, Rochester, MN, in 1988 after receiving a BSEE with a Computer Engineering option from Michigan State University.

Teresa C. Kan
Ms. Kan is a staff programmer of the database quality and performance team in Rochester current systems. She joined IBM, Rochester, MN, in 1984 and was responsible for DDM development and support for System/36. In 1986, she joined the AS/400 DDM team and was responsible for the byte stream model development. In the past two years, Ms. Kan was the design focal point for the Distributed Relational Database Architecture protocol on the AS/400 system. She was the AS/400 representative on the DRDA1 Control Board. She received an MS degree in Computer Science from Utah State University, Logan, UT.

Dennis L. Karst
Mr. Karst is an advisory engineer in interconnect technology working on fiber-optic data link development. Past assignments include electromagnetic compatibility and optical disk storage development. He holds three US patents and has 10 inventions published.
Mr. Karst joined IBM, Rochester, MN, in 1980 and received a BSEE from South Dakota State University, Brookings, SD, in 1979 and an MSEE from the University of Minnesota, Minneapolis, MN, in 1986.

Gary L. Kearns
Mr. Kearns is an advisory engineer in systems manufacturing test support. His current activities include development and implementation of stress testing and guardband performance analysis techniques. He also supports process data collection and information analysis for process improvement initiatives. He supported new product early entry activities with quality planning and test requirements definition for System/3 card handling machines, optical character readers, and the 5100 computer. He supported VLSI component evaluation and failure rate forecasting while working in the component quality and reliability lab. He facilitated the introduction of low cost system stress screen processes for the 5360, 5362, 5363, 5364, and System/38 computers. Mr. Kearns joined IBM, Rochester, MN, after receiving a BSME from the University of Minnesota.

Kevin A. Kelle
Mr. Kelle is a staff programmer in the system supervisor area of VLIC programming. Prior to this, he was a member of the AS/400 screen team. Past assignments have been in disk storage management. On the AS/400 system, he worked on the mirroring and checksum projects and also contributed to the design of the I/O subsystem. On the System/36, he developed disk microcode. Mr. Kelle joined IBM, Rochester, MN, in 1984 after he received a BS degree in Computer Science from the University of Nebraska, College of Engineering and Technology, Lincoln, NE.

Harvey G. Kiel
Mr. Kiel is an advisory engineer in work station controller development. He is responsible for work station controller architecture. Previous

assignments include work station controller team leader and development on the 5294 Remote Work Station Controller and 5260 Retail System. He filed 15 patents and published 38 disclosures. Mr. Kiel joined IBM in 1978 after receiving a BSEE from South Dakota State University, Brookings, SD.

Jeffrey C. Kramer
Mr. Kramer is an advisory programmer in the system management design control group. He joined IBM in Rochester, MN, in 1983 and has held various assignments on System/38 and the AS/400 system. His past AS/400 assignments include design and development of service processor licensed internal code, APPN, communications I/O interfaces for heterogeneous protocols, and OSI. Mr. Kramer holds a BS in Computer Science and Mathematics from North Dakota State University, Fargo, ND.

Timothy L. Kramer
Mr. Kramer is an advisory programmer working in PC Support/400 installation and configuration. Prior to working in the PC Support area, he developed System/36 and AS/400 office products. Mr. Kramer joined IBM, Rochester, MN, in 1979 after teaching at the secondary school level. He graduated from St. Olaf College, Northfield, MN, with a BA in Mathematics.

Janet H. Krueger
Ms. Krueger is the lead designer for the PC Support/400 product. She is a strategist for future cooperative processing support on the AS/400 system and the Rochester representative on the SAA Client/Server council. She joined IBM, Rochester, MN, in 1976, working in programming language development for the System/34. Later, she transferred to the programming support area, designing and coding both MVS and VM application development tools for internal IBM use. Ms. Krueger has a BA in German and Business from Central College, Pella, IA, and an MS in Computer Science from the University of Iowa, Iowa City, IA.

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.

Sheldon B. Levenstein
Mr. Levenstein is a staff engineer in AS/400 9406 processor development. He joined IBM in 1980 as a junior engineer. Mr. Levenstein is involved in the development of the AS/400 processor chips. He had several assignments dealing with the processor, storage control, and multiprocessor design. He applied for four patents and was awarded the outstanding technical achievement award for his contributions to the AS/400 multiprocessor effort. Mr. Levenstein received a BSEE from the University of Illinois in 1980.

Dennis R. Martin
Dr. Martin is a staff programmer in VLIC storage management, with both architectural and development responsibilities in AS/400 availability and auxiliary storage management. He came to IBM in 1987, after 11 years of college and university teaching, to become a member of the team that designed and developed disk mirroring for the AS/400 system. Dr. Martin continues as a member of the adjunct faculty in computer science at Winona State University, MN, and has two US patents and several disclosures published since coming to IBM. He holds a PhD in Musicology from the University of Iowa.

Timothy J. Massaro
Mr. Massaro is an advisory programmer working as team leader of the Operational Assistant programming team. He joined IBM in Rochester, MN, in 1980. He has a BS degree in Computer Science with a minor in Mathematics from the University of North Dakota, Grand Forks, ND. He held numerous programming responsibilities including the System/38 work management team and moving the message handler and menu components to the AS/400 system. He received the IBM invention award for two US patents.

Michael J. McDermott
Mr. McDermott is an advisory programmer in storage management for the AS/400 system. He is responsible for the technical leadership of the storage management component and the system availability concerns relating to disk. Past IBM assignments include design and development of supervisor microcode on the System/36 and system software for System/34, System/32, and System/3. He has 11 invention disclosures published and three patents filed. He graduated from Iowa State University with a BS in Mathematics.

John E. McGinn
Mr. McGinn is a staff programmer in AS/400 communications development. He has been the team leader of the APPN project for the past two years. His previous assignments include design and code development for the topology and routing services (TRS) component and the machine services control point (MSCP) component on the AS/400 system. He also was involved in the System/36 APPN project. He holds two US patents and has two technical disclosures published. He joined IBM, Rochester, MN, in 1984 after receiving his BS degree in Computer Science from the University of Wisconsin at LaCrosse.

Lynn A. McMahon
Mr. McMahon is an advisory programmer in AS/400 HLIC development. He joined IBM in 1977 as an associate programmer and worked on various emulation and device control microcode projects before becoming involved in horizontal licensed internal code development for the System/38 and AS/400 processors. Mr. McMahon received two patents, applied for four others, and received an outstanding technical achievement award for his work on the AS/400 multiprocessor IMPI definition. He received a BS degree in Computer Science from Iowa State University in 1975 and an MS degree in Computer Science from Iowa State University in 1977.

James R. Morcomb
Mr. Morcomb is a senior programmer and manager of SystemView strategy and architecture. He joined IBM in 1957 and held various development and development management positions with emphasis in reliability, availability, service, customer support, and system management. He managed the engineering RAS development department on the System/38 with primary responsibility for developing the advanced service support capability of the system. From 1982 to 1985 he worked on strategy and architecture for future systems. That work became the foundation for electronic customer support and much of the system management on the AS/400 system. His current responsibilities include SystemView strategy and architecture for the AS/400 system, representing ABS on corporate SystemView councils and activities, and managing a cross-line-of-business team with responsibility for IBM-wide implementation of electronic customer support.

Timothy J. Mullins
Mr. Mullins is an advisory engineer in the hardware design center and is involved in AS/400 system design and performance analysis. He worked in the design and development of I/O controllers for System/38 user display devices, such as work stations and the operator console. Mr. Mullins later became involved in central processing unit development in the areas of logic design and timing analysis. Mr. Mullins joined the Rochester Engineering Laboratory 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.

Thomas N. Paske
Mr. Paske is a senior engineer. He is currently responsible for development and implementation of manufacturing strategies for the Rochester systems plant. Previous assignments include the introduction of new products into manufacturing, including card I/O machines, System/36, System/38, and the AS/400 system. He joined IBM, Rochester, MN, in 1961.

Ricky M. Peterson
Mr. Peterson is a staff engineer in the AS/400 performance group. He previously worked on System/36 and AS/400 work station controller performance analysis and was team leader for 9404 performance analysis. He has one patent, six technical disclosures, and three technical papers published. Mr. Peterson joined IBM, Rochester, MN, in 1984 after receiving a BS in Computer Engineering from Iowa State University.

Patrick T. Priniski
Mr. Priniski is a staff programmer with design and development responsibilities in the remote SQL function of PC Support. He joined IBM in Poughkeepsie, NY, in 1982 after earning a Bachelor of Music Education degree and a BS degree in Data Processing from Northern Michigan University, Marquette, MI. Since transferring to Rochester in 1988, he has been team leader for several PC Support functions.

Kevin J. Przybylski
Mr. Przybylski is a staff engineer working on product EMC design and evaluation. His current assignment is to develop and implement new EMC strategies and designs for use in future IBM systems. Past assignments include EMC design for the AS/400 system and System/38. Mr. Przybylski joined IBM, Rochester, MN, in 1981 after receiving his BSEE from the Milwaukee School of Engineering, Milwaukee, WI.

Carol L. Ramler
Ms. Ramler is a senior associate programmer in the SQL/400 development group. She is the team leader for the SQL precompilers and parsers. Ms. Ramler joined IBM, Rochester, MN, in 1987 after working as a programmer for Control Data Corporation. She has a BS degree in Computer Science from Saint Cloud State University, St. Cloud, MN.

Julie A. Ransom
Ms. Ransom is a senior product planner in the advanced systems planning area. She joined IBM in Rochester, MN, in 1978. She held numerous programming responsibilities including writing applications to support the Rochester site, programming of robotics for the manufacturing of disk products, and working with business partners and customers that migrated applications to the AS/400 system prior to its announcement in 1988. Since then, she has worked in the product planning area focusing on user interfaces. She has a BS degree in Computer Science and Mathematics from Mankato State University, Mankato, MN.

Paula H. Richards
Ms. Richards is a staff information developer in OS/400 programming development. Ms. Richards is currently working on graphical user interface enabling for the OS/400 operating system. Past IBM assignments include technical design and leadership roles for the AS/400 hypertext and CD-ROM softcopy projects. Her work experiences prior to joining IBM in 1987 include working for a small systems integration company designing database applications for personal computers. She received her BS and MA in education from Western Carolina University, NC.

Joseph T. Rommel
Mr. Rommel is a staff industrial engineer. He currently supports manufacturing and distribution operations as part of the business planning department. He worked on the planning and implementation stages of the AS/400 production facility. He joined IBM at Rochester, MN, in 1984 after receiving a BSIE from Iowa State University.

Bruce G. Rudolph
Mr. Rudolph is a staff engineer for circuit technology in AS/400 systems logic. He is responsible for system clock generation for the AS/400 processor, main storage controller, and I/O interface and control units. Past assignments include bipolar and CMOS circuit design, chip logic design, technology application support for the processor development area, and the design of clock trees and clocking methodologies for use on ASIC chips. He is credited with two US patents and four invention disclosures in the field of clock generation for ASIC designs. He joined IBM, Rochester, MN, in 1983 after he received a BSEE degree from the University of Wisconsin, Milwaukee, WI.

Richard A. Saltness
Mr. Saltness is a staff engineer and is a member of the CPPC data management team. Past IBM assignments include the development of software for the AS/400 functional testers and logic card tester development. He joined IBM, Rochester, MN, in 1984 after he received a BSEE degree from the University of Wisconsin, Madison, WI.

Phillip C. Schloss
Mr. Schloss is an advisory engineer working on the PC Support tools folder and tools for PC Support programming. He joined IBM in 1965 as a system engineer. In 1969, he transferred to Rochester, MN, to design hardware and microcode for work stations. Later, he transferred to the Rochester Programming Laboratory to design and code for the AS/400 PC Support product. Mr. Schloss has a BS degree in Electrical Engineering from North Dakota State University, Fargo, ND.

Dennis J. Schmidt
Mr. Schmidt is an advisory programmer working on user interface design and development for AS/400 and future systems. He held numerous programming responsibilities including the System/36 main storage supervisor, System/36 security, AS/400 System/36 Migration Aid, AS/400 System/38 Migration Aid, AS/400 work management, and AS/400 Operational Assistant interface. He received the IBM invention award for two US patents. He joined IBM in Rochester, MN, in 1980. Mr. Schmidt has a BS degree in Applied Mathematics from the University of Wisconsin-Stout, Menominee, WI.

Randall A. Schmidt
Mr. Schmidt is an engineer in design systems. He works with the development and support of automated logic synthesis of VHDL chip designs for the AS/400 system. He joined IBM, Rochester, MN, in 1988 upon receiving a BSEE degree from St. Cloud State University, St. Cloud, MN.

Quentin G. Schmierer
Mr. Schmierer is a senior engineer responsible for VHDL language and logic synthesis tools development for the AS/400 processor group. He is credited with five patents and numerous disclosures for work on the original AS/400 processor and System/38 I/O adapters. He received two outstanding technical achievement awards for work on the VHDL macro processor program and work on the AS/400 storage control unit hardware. He joined IBM in 1976 and holds a BSEE degree from North Dakota State University in Fargo, ND.

Clark A. Scholten
Mr. Scholten is a senior associate programmer in client API development. Currently, he is the team leader of the data queues function of PC Support. Mr. Scholten also worked on the PC Support message function and the PC Support configuration process. He joined IBM in 1987. Mr. Scholten has a BA degree in Computer Science and Mathematics from Northwestern College, Orange City, IA.

Michael D. Seyfert
Mr. Seyfert is a staff engineer responsible for the mechanical hardware design of card enclosures and frames for future high-end systems. Past IBM assignments include manufacturing support of flexible circuits for disk drives, process development for surface-mount card assemblies, and product development of both low- and high-end AS/400 systems. He joined advanced systems development in 1987. He holds a patent for a card enclosure construction method, and has published five technical disclosures. Mr. Seyfert joined IBM, Rochester, MN, in 1983 shortly after receiving his BSME degree from the University of Wisconsin at Madison.

Charles C. Shih
Mr. Shih is the development manager for the OSI Communications Subsystem/400 licensed program in Palo Alto, CA. He worked for 6 years in Rochester, MN, on 3270 emulation and X.25 support for the System/38, and APPN and common I/O support on the AS/400 system. He holds a BS in Computer Engineering from the University of Illinois.

David W. Siljenberg
Mr. Siljenberg is an advisory engineer in the circuit technology group. He is the lead engineer for the fiber-optic link card electronics. Past IBM assignments include the design of analog integrated circuits for disk drives. Mr. Siljenberg joined IBM, Rochester, MN, in 1979 after working for Motorola, Schaumburg, IL. He received his BSEE from the Illinois Institute of Technology, Chicago, IL, and his MSEE from the University of Illinois, Champaign, IL.

Ronald L. Soderstrom
Dr. Soderstrom is a senior engineer in interconnect technology working on fiber-optic data link development. Past development assignments were on optical character recognition (OCR) scanners, thermal printing systems, laser supermarket scanners, and optical disk storage systems. He holds 11 US patents and has 21 inventions published. Dr. Soderstrom joined IBM, Rochester, MN, in 1966. He received a BSEE in 1964, an MSEE in 1966, and a PhD (EE) in 1972, all from the University of Minnesota, Minneapolis, MN.

Frank G. Soltis
Dr. Soltis is a senior engineer in AS/400 system architecture. He joined IBM in 1963 and has held numerous development and development management positions. In 1968 he received his PhD from Iowa State University in Ames, IA. He was an original architect of the System/38 and AS/400 architectures and is currently leading the effort to define the AS/400 architecture for the late 1990s. He holds 12 patents, 18 disclosures and is an adjunct professor of Electrical Engineering at the University of Minnesota.

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.

Gary L. Stowe
Mr. Stowe is a staff engineer responsible for the assembly/test process of the high-end AS/400 systems. He supports all aspects of manufacturing the AS/400 systems. He joined IBM, Rochester, MN, in 1984 and held positions in release, process, and packaging engineering. He received a BS in Mechanical Engineering in 1984.

Jeffrey W. Tenner
Mr. Tenner is an advisory programmer in the database management area. He is the team leader for the distributed database function. Before joining the database management area in 1987, he developed program development

applications for MVS and VM environments and tested a variety of System/38 functions. He has two disclosures published. Mr. Tenner joined IBM, Rochester, MN, in 1981 after he received a BS degree in Computer Science from the University of Wisconsin, LaCrosse, WI.

Mark M. Thornton
Mr. Thornton is a staff engineer in mechanical development; he is currently involved in mechanical design of the AS/400 line of IBM computers. He joined IBM in 1982 at Manassas, VA, in the Federal Systems Division. From 1982 until 1988, he worked in the area of thermal analysis and test and participated in government sponsored Industrial Research and Development (IRAD). From 1988 until 1991, he worked on developing the mechanical hardware for the Version 2 AS/400 system. Mr. Thornton received a BSME from Iowa State University, Ames, IA, in 1981 and an MSME from the University of Maryland, College Park, MD, in 1986.

Scott M. Thorvilson
Mr. Thorvilson is a senior associate engineer in systems packaging, working in EMC design and evaluation. His current assignment is EMC ownership for all models of the 9406 and research into conductive plastics for EMC applications. Mr. Thorvilson joined IBM, Rochester, MN, in 1988 after receiving his BSEE degree from Marquette University, Milwaukee, WI.

James R. Ulwelling
Mr. Ulwelling is a senior associate programmer in the programming enabling interfaces area. He is currently the team leader of DDS compiler development. He has been a member of the DDS compiler team since 1986. He has three disclosures published. Mr. Ulwelling joined IBM, Rochester, MN, in 1984 as a drafting technician after receiving his mechanical drafter diploma from Rochester Area Vocational Technical Institute, Rochester, MN. He moved into programming in 1986 and has since been actively pursuing a BS degree in Computer Science at Winona State University, Winona, MN.

David A. Wall
Mr. Wall is a staff programmer in the client development area at IBM, Rochester, MN. He currently is the team leader of the shared folders function group. He has also worked in the client development area of systems assurance. He joined IBM in 1984 after graduating from the University of Nebraska, Lincoln, NE, with a BS degree in Computer Science.

Mark G. Wenzel
Mr. Wenzel is a senior associate programmer in the client environments area. He is currently working with memory management issues for PC Support. Previously, he worked on the virtual print, work station function, and text assist functions of PC Support. Mr. Wenzel joined IBM, Rochester, MN, in 1988 after graduating from Augustana College, Sioux Falls, SD, with a BA in Business Administration and Computer Science.

Stephen E. Wheeler
Mr. Wheeler is an advisory engineer in the systems manufacturing process and strategy development group. He is currently engaged in early manufacturing involvement on future systems development. He holds eight patents and has 50 published inventions. He joined IBM in 1967 after receiving a BSME degree from the University of Wisconsin at Madison. Mr. Wheeler has worked in many development and manufacturing projects.

Robert R. Williams
Mr. Williams is a senior engineer responsible for interchip communications. Past IBM assignments include the development of chip images, delay calculation, power assignment, and circuit design. He has seven patents and 10 published disclosures. Mr. Williams joined IBM, Rochester, MN, in 1968 and holds BSEE and MSEE degrees from North Dakota State University, Fargo, ND.

Andrew H. Wottreng
Mr. Wottreng is a senior engineer in the area of AS/400 high-end processor architecture and definition. He joined IBM in 1974 as a junior engineer. Mr. Wottreng has been involved in the development of System/34, System/36, and AS/400 processing units. He has had several assignments dealing with chip designs, performance modeling, maintenance procedures, and architecture. He has received one patent, applied for one other, and received an outstanding technical achievement award for his work on the AS/400 multiprocessor architecture. Mr. Wottreng received a BSEE from Marquette University in 1971 and an MSEE from the University of Minnesota in 1980.

Mark G. Wulf
Mr. Wulf is a senior associate programmer working on the development of distributed relational database. He joined IBM in 1988 and worked on distributed data management before moving to distributed relational data-

base in 1990. Mr. Wulf received a BS in Computer Science in 1988 from the University of Kansas.

S325-6020-00

Printed in U.S.A.