IBM Systems Application Architecture (SAA)

In this report:

Synopsis

Editor's Note

SAA development intensifies every year. The introduction of System/390 in September 1990 upgraded all mainframe environments and added many new SAA elements—only one month after expansion of OS/400 midrange SAA offerings. In March 1991, APPN networking was designated a key CCS element, with provisions for OS/2 EE as a powerful Network Node. Significant architectural extensions include the Print Manager Interface, the Resource Recovery Interface, Distributed Relational Data Access (DRDA), and the SystemView management architecture.

Report Highlights

SAA is IBM's most ambitious architectural undertaking, because it attempts to organize many types of compatibility and then implement full compliance in six leading operating environments (MVS/TSO, MVS/CICS, MVS/IMS TM, VM/CMS, OS/400, and OS/2 EE) and partial compliance in several others (VSE, AIX, etc.). In order for programs to be portable and distributable, each environment must have corresponding languages and services while upgrading or creating an enormous number of software products.

There are now two areas of intense activity:

- Expansion of the SAA model through auxiliary architectures.
- A constant flow of new products that supply missing elements of the basic architecture.

This report provides an introduction to SAA. It also provides an overall picture of the expansion of the SAA model and tracks how the various products supplied in each environment reflect the level of overall development. Understanding what SAA is meant to do in the final analysis is not so difficult. Understanding how it stacks up right now relative to its goals will probably never be simple, because of the number of programs, standards, and architectures it is controlling.

This report was prepared exclusively for Datapro by Elinor Gebremedhin, an independent data communications consultant and freelance writer. Ms. Gebremedhin possesses a wide background in IBM midrange and mainframe systems; her current areas of focus include IBM and IBM-compatible systems, SAA, distributed processing, and communications software.

Analysis

What Drives SAA?

SAA is advertised as IBM's developing standard for architecting solutions to compatibility problems in its major environments. User interest is driven by two factors—the need to develop new types of processing (cooperative and distributed) and the need for solutions to problems created by mergers of new companies or historically independent operating divisions, or other situations which lead to the need for interaction between incompatible environments. In addition, as IBM develops SAA, and users begin to use it in a variety of contexts, new needs appear.

Processing Goals

Initially, SAA development must pass through four processes in order to reach its

goals. Currently, IBM is focused at architecting the third level, implementing the second level, and putting the finishing touches to the first level. These levels are fundamentally hierarchical because the solution to one type of processing problem/user need has been an essential foundation for work on the next level. The four types of processes involved are as follows.

Porting of Standalone Programs: Porting programs involves moving them from one IBM environment to another or converting to a new version of an operating system that includes substantive changes affecting the application programs. A communications subsystem need not be involved. When the user base involved has been large enough, the time-honored solution has been assistance by special converters, usually developed by third-party vendors. The most recent example involves the upgrading of System/36 applications for the AS/400. The SAA solution is to standardize high-level language compilers, data formats, and supporting services to the extent that the program can be recompiled and run. Where the fundamental system architecture prevents this solution from being reached, differences are carefully documented so that users can tailor individual applications.

Cooperative Processing: Cooperative programs split functions between a front-end workstation and a back-end processor. The front end usually supports an end-user interface that supports graphics, other ease-of-use features, and perhaps some of the processing. The rest of the application, plus some traditional server functions, runs in the host. This differs from pure client/server computing, where the client does nearly all of the application processing and the server usually performs application-independent services such as communications, data storage and management, and backup/restore functions.

Since the front-end and back-end processors are likely to be different architectures, SAA portability solutions have something to offer. In addition, front-end workstations can be attached to hosts through LANs as well as direct connections, so SAA definitions of the relationship and management of LANs become important. IBM sees cooperative processing, which is viable now, as the trend of the 1990s. Most of the programs providing services for the Applications Development/Cycle (AD/Cycle) architecture are themselves cooperative, requiring a PS/2 workstation to interface to the host or server.

Distributed Databases: Databases are distributed when an end user can access data automatically from one of several possible locations in a network, without the end user knowing which database served the request. There are a number of levels of definition for this capability, such as bringing database segments to the end user to work on locally, or doing the work on the remote system and bringing the results to the end user. The requisite database logic has become available for mainframes and OS/2 EE systems, but not the OS/400. The state of the art in IBM SNA mainframe peer-to-peer networking imposes restrictions on flexibility, integrity, and performance. Ironically, midrange APPN networks are much more suited to support of distributed databases, but only PS/2s with the new Networking/2 and distributed database software can take full advantage at this point in time.

Distributed Processing: Distributed processing provides for locating (and relocating) programs or program segments on different systems transparently, so that the end user need not know where the program is or whether its location has changed. Distributed processing is a step up from the distribution of data, because it implies program portability; a system for tracking and managing the programs or program segments; and, as with distributed databases, dynamic addressing and rerouting in the underlying communications subsystem. In addition to the networking issues in common with data distribution, it has become clear that distributed processing will not work well without a higher level of system and network automation than is currently available.

What has made users so interested in the newer forms of processing? Why not stick with the tried and true? Some factors driving this development are forward looking, like the interest in better sharing of data and in supplying more timely analyses from more reliable sources. The more processors linked together to swap programs and data, the more likely it is that processors from multiple architectures will participate. Nonproprietary architectures like OSI or UNIX become more attractive, even if individual implementations are not perfectly compatible with one another. This is especially true for the more than half of IBM's user base that lives outside of the United States.

These factors tend to affect new users or existing users of smaller systems which have a small enough base that a conversion is not a financial disaster.

Immediate Problems

The drive toward better program portability and subsystem compatibility arises not only from changes in the types of applications being developed. One of the most common reasons is an immediate practical need to merge systems from two incompatible networks as the result of a company merger. This can affect larger systems because the computers inherited from the merged company may be highly incompatible, yet need to interact with one another.

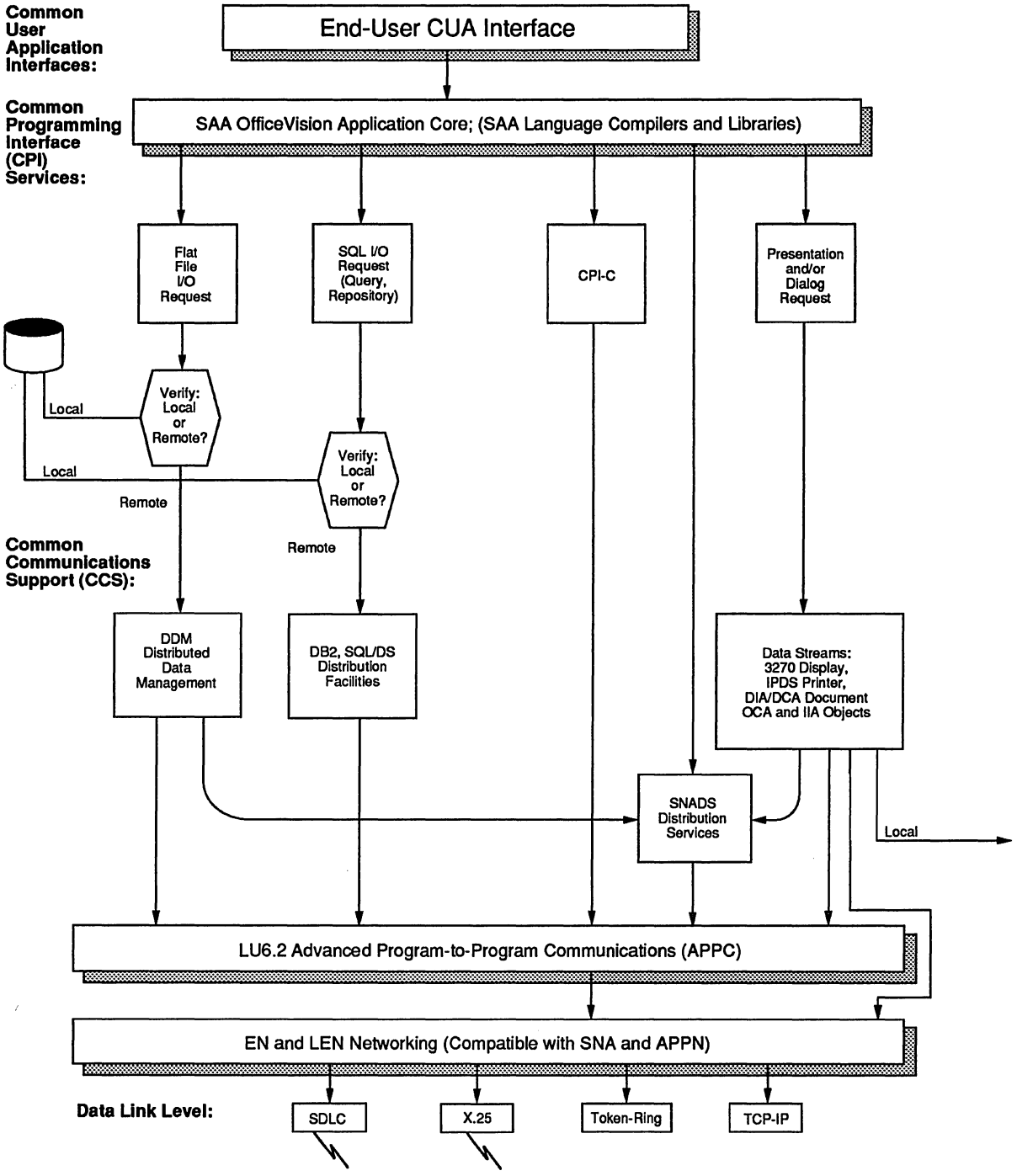
Companies that ordinarily would not want to pay the cost of converting find themselves forced to do so in order to run an integrated business. In addition, governments the world over are interested in networking a variety of system types, talking to one another across national boundaries, and swapping data at a variety of levels. Many of them, including the United States government, have settled on OSI networking rather than favor one vendor over another. As of August 1990, the U.S. government has stipulated that new networking acquisitions must conform to OSI specifications.

This decision is undoubtedly part of the reason IBM has been driven into incorporating the upper levels of OSI into SAA.

Specific Architectural Goals

The SAA architecture defines specific characteristics and international standards to be used for SAA *elements* such as high-level language compilers; their supporting service programs and databases; supporting communications network and data flows; development and testing tools; end-user interface formats; and all kinds of management systems, production cycles, program development cycles, and networking. These are arranged in a core hierarchy consisting of Common User Access (CUA) standards, Common

Figure 1.
SAA Flows toward SNA



OfficeVision applications typically call on the full range of CPI and CCS services and even SNA networking facilities not yet part of SAA.

Programming Interface (CPI) standards, Common Communications Support (CCS) standards, National Language standards, and adjunct architectures such as AD/Cycle. Figure 1 shows how a comprehensive communications application like IBM's OfficeVision can involve most or all of the core standards.

SAA promises that eventually every SAA core *element* will be available for every SAA *environment.* Naturally, the user does not need to buy it all to create an SAA-compliant program. Users need to be careful to implement the appropriate corresponding subsets on each system in order to set the stage for the desired levels of program portability and distributability.

SAA works because elements (languages, services, etc.) on all systems have firm protocol boundaries to separate and/or translate the system's unique characteristics into SAA functions and characteristics. The end users, applications programmers, and to some extent systems programmers are presented with high-level system-independent interfaces. They will view the same types of screens, create programs using the same instruction set, call on the same supporting functions, and invoke printing in the same way. They deal with the same "shell," no matter which of the SAA computing environments are being used.

Ultimately, all SAA elements will have the corresponding Application Programming Interfaces (APIs) in each environment and open, detailed source code descriptions allowing development of portable third-party or end-user programs.

IBM as a Multivendor

At this point, it is probably obvious that even the basic SAA core impacts an enormous number of program products. All the existing Cobol, Fortran, PL/1, and other SAA compilers need to be adjusted; new compilers need to be added; database software regulated; and communications software developed. What put IBM into the position where it was willing to take on these tasks?

IBM may have been the vendor that introduced the idea of upward compatibility with System/360 and System/370, but when the minicomputer market gathered impetus in the mid-1970s, IBM found that it did not compete well for certain types of applications. IBM management felt that the company had gotten too big, so it set up several independent divisions to compete with each other as a way to compete with other vendors like Digital. System/36 and System/38 were the result, as well as Series/1 and several other less successful products.

Later, for different reasons, IBM PCs were engineered from off-the-shelf components. The result: development money was spread too thin. There were so many product lines that IBM looked like a vendor that had gone through a series of mergers. The amalgamation of System/36, System/38, 8100, and 5100 product lines into the AS/400 helped, but it was not good enough to satisfy all the compatibility requirements of IBM customers and IBM itself. SAA was introduced on March 17, 1987, a year before the AS/400 was announced.

The company that needs SAA the most may be IBM. Once the foundations exist, software development will be more cost effective because of the ability to port programs from one system to another.

New Needs

Ideally, the use of SAA high-level language compilers would mean that the same source code could be compiled on different systems and run on each of them without need for further alteration. This would mean that once distributed networking was implemented, changes in work load or in system or network availability could result in an automatic relocation of that program to a system that was SAA compatible, but not necessarily compatible at the machine level.

In actuality, as we will see later on, there are a number of ways in which SAA compilers, at their present level of development, can cause unexpected or unreliable results when the same instructions are recompiled on another system. SAA documentation openly describes these problems so that users can adjust applications accordingly. Furthermore, compatibility can be compromised if a program uses interlanguage calls or calls on a service in one environment that are not supported in another. Thus, SAA drives IBM not only to implement the *same* programs in each environment but also to upgrade those programs as new ways are developed to smooth out fundamental system incompatibilities.

As IBM users try these system and processing relationships, new needs develop, driving IBM toward new SAA adjunct architectures and supporting program products. Two examples of reactive announcements include:

- AConnS Application Connectivity Services, for supporting connection of cooperative workstations.
- SystemView, a management architecture combining system and network management under one umbrella.

As we investigate SAA in more depth, the reasons for these new needs will become more apparent.

Environments and Platforms

The core of SAA development revolves around IBM's commitment to implement a full set of standard functions or elements in each of the designated SAA software environments.

Regardless of whether it works, the concept itself should be simple enough to understand. As it turns out, however, grasping how it is to be implemented is complicated by four factors. First, as key environments have evolved, IBM names for the latest version have not always been the best for distinguishing an entity from its forebears and associates. Second, IBM and its user base have evolved a veritable alphabet soup of acronyms that are recognized by insiders in each environment as the *real* names, but are unintelligible to outsiders. Third, users sometimes begin to use historically specific acronyms loosely or generically to indicate several later versions. Fourth, the distinction between element and environment is not really apparent for CICS, which is treated as primary under MVS but as subordinate elsewhere.

Table 1 untangles these identification problems by presenting the evolution of the formal names of the SAA-related mainframe operating systems, the acronyms that users recognize as their real names, and the most likely alternate designations.

The alternates are either designated by IBM (as in MVS/SP Versions 1, 2, 3, and 4) or have come to be used in a loose generic sense to include more than one operating system version (as in VM/CMS). In most cases, understanding the unique identity of a particular version is best grasped by associating it with its primary platform, rather than all the different platforms it runs on.

The esoteric quality of the language of IBM'ers is largely due to their gravitation toward the use of acronyms, to the point where recognition of the original name of a piece of software is practically lost (MVS is a case in point). This is not only a problem for outsiders: people thoroughly familiar with the language of their own IBM environment will have problems with another—a situation that should improve as a side effect of SAA! The changing usages of acronyms in each is also a problem area, making the "historical" data of Table 1 surprisingly important to the understanding of current usage. A midrange user steeped in the relationships between Systems/36, /38, and the AS/400 may hear a mainframe user complaining today about

Table 1. The Evolution of IBM Mainframe Environment Designations

Key Acronym	Full Name	Alternate Usage*	Primary Platforms	Other Platforms**
Most Current Generics				
MVS	Multiple Virtual System	OS	All	All
VM	Virtual Machine	VM/CMS	All	All
VSE	Virtual System Extended	DOS	All	All
DOS and VSE Environments				
DOS	Disk Operating System	—	System/360	System/370
DOS/VS	DOS/Virtual System	DOS	System/370	3030
DOS/VSE	DOS/Virtual System Extended	VSE	4300	3030, 3080, 3090, 9370
VSE/SP	VSE/System Product	VSE	9370, 4381	3080, 3090,
VSE/ESA	VSE/Enterprise Systems Architecture	VSE	9000	3090, 4381
OS and MVS Environments				
OS	Operating System	—	System/360	System/370
OS/VS	OS/Virtual Storage	OS	System/370	—
SVS	Single Virtual Storage	—	System/370	—
MVS	Multiple Virtual Storage	—	System/370	3030
MVS/SE	MVS/System Extended	MVS/370	3030	3080
MVS/SP	MVS/System Product	MVS	3080, 4300	3090
MVS/370	MVS/370 Architecture	MVS/SP Version 1 (& MVS/SE)	3080, 4300 (& 3030)	3090
MVS/XA	MVS/Extended Addressing	MVS/SP Version 2	3080, 4381	3090, 3080, 9000
MVS/ESA	MVS/Enterprise Systems Architecture	MVS/SP Version 3	3090, 4381	3080, 9000
MVS/ESA SP	MVS/ESA System Product	MVS/SP Version 4	ES/9000; 3090-9000T	3080, 3090, 4381
VM/CMS Environments				
VM/SP	VM/System Product	VM	4300	9370, 3080, 3090, 9000
VM/IS	VM/Integrated System	VM	9370	4300, 3080, 3090, 9000
HPO	High Performance Option	—	4381	3080, 3090, 9000
VM/XA SP	VM/Extended Addressing System Product	VM/XA	3080	4381, 3090, 9000
VM/ESA	VM/Enterprise Systems Architecture	VM	9000	4381, 3090

**Not exhaustive; only the most likely alternative is listed.*
***"Other Platforms" excludes guest operating systems running under VM.*

having to do a DOS-to-OS conversion. The midrange user thinks this is referencing something about old System/360 systems, or perhaps even PCs, when actually what is going on is a change from VSE to MVS/ESA on a 4381 or 3090.

Environments

The SAA environments that are currently earmarked for complete development are as follows.

TSO/E: This is the Time Sharing Option/Extended subsystem, running in conjunction with the MVS/ESA SP operating system. TSO/E is a straightforward environment comparable to the CMS component of VM.

CICS/ESA: This is the Customer Information Communication System/ESA transaction processing subsystem, running in conjunction with MVS/ESA SP. The earlier CICS/MVS version is also supported.

IMS/ESA TM: The Information Management System/ESA Transaction Manager, running in conjunction with the MVS/ESA SP, is now a standalone product that can access either DB2 or IMS databases. Its predecessor, IMS/VS DC (Data Communications), required that the IMS database be present. Both are currently supported as SAA environments. The close association of DC and IMS/VS leads one to believe that IMS itself is an SAA standard, but it is not.

Table 2. SAA Transaction Processing

Host Environment	MVS/ESA TSO/E	MVS/ESA CICS/ESA	MVS/ESA IMB/VS TM	VM/CMS	OS/400	OS/2 EE	VSE/CICS
SAA Support	•	•	•	•	•	•	P
Interactive Applications	•	—	—	•	•	•	—
Transaction Processing *(2)*	—	•	•	*(1)*	•	•	•
Cooperative Workstations:							
OS/2 EE	•	•	•	•	•	•	•
CICS OS/2	—	•	—	*(1)*	—	—	•
Resource Recovery Interface (RRI)	—	—	•	•	—	—	—
Extended Recovery Facilities (XRF)	—	•	•	—	—	—	•

•=Supported.
P=Partial support.
—=Not supported yet.
(1) VM/CMS itself is an interactive timesharing system. However, there are non-SAA VM versions of CICS that can turn VM into a transaction processing environment. SAA ignores them, so this table does too.
(2) TPF2, IBM's most powerful transaction processing program, is not included because its participation in SAA is minimal at present.

Acronyms:
CICS—Customer Information Control System.
CMS—Conversational Monitor System.
EE—Extended Edition.
ESA—Enterprise Systems Architecture.
IMS—Information Management System.
MVS—Multiple Virtual Storage.
TM—Transaction Manager.
TSO/E—Time Sharing Option/Extended.
VM—Virtual Machine.
VS—Virtual Storage.
VSE—Virtual Storage Extended.

VM/CMS: The Virtual Machine/Conversational Monitor System, in its VM/ESA version, has replaced the capabilities of both VM/SP and VM/XA. IBM has designated VM/CMS as the front-running SAA VM environment for SAA but continues to support VM/SP. VM/XA has been dropped. VM systems are not only interactive timesharing systems in their own right but are also capable of running other operating systems under them, each in separate partitions, but with shared I/O and memory facilities. VM/SP must operate in conjunction with HPO on many processor models and all multiprocessor systems.

OS/400: The Operating System/400 multipurpose midrange operating system is closely tied to AS/400 hardware and firmware and is unlikely to admit additional non-SAA environments.

OS/2 EE: This refers to the Operating System/2 Extended Edition for PS/2 programmable workstations. Other PS/2 environments, such as OS/2 SE and DOS, are not included as full SAA environments even though they can run some SAA software elements.

In addition, two environments have been specially earmarked for development as distributed systems, with partial implementation of SAA interfaces. They are:

VSE/ESA CICS: This is the current DOS version of CICS running under Virtual System Extended/ESA, which can develop and exchange programs with CICS systems running under VM and OS/2 EE as well as MVS.

AIX: AIX refers to the Advanced Interactive Executive, particularly the version on the RISC/6000 series. AIX will eventually have a CICS interface, as well as the current set of lower level SAA elements. Because of the widespread acceptance of UNIX as a nonproprietary architecture, AIX is discussed together with OSI at the end of this report.

When SAA was announced on March 17, 1987, it was targeted at specific core environments; the existing mainframe MVS/XA, TSO/E, and VM/CMS (VM/SP or VM/XA) operating environments; the then newly introduced PS/2 systems when running under OS/2 Extended Edition; and the anticipated "Silverlake" follow-on to the S/3X minicomputers. Within the next four years, SAA evolved together with dramatic changes at all levels except for the new OS/2. Key events were as follows.

- March 1987: MVS/XA TSO, VM/CMS (VM/SP, VM/XA), and OS/2 EE were designated as SAA environments.
- October 1987: IMS/VS DC and CICS/MVS environments were added, also to run under MVS/XA.
- March 1988: MVS/ESA was introduced as an upgrade to MVS/XA and was designated as the SAA operating system for the TSO/E, CICS/MVS, and IMS/VS DC environments.
- June 1988: The AS/400 (Silverlake) product line was introduced as an SAA environment, with many SAA elements bundled into the OS/400 operating system.

- September 1988: A Statement of Direction was issued formulating an SAA-compatible DOS/VSE CICS subset of the full SAA facilities. VSE SAA will be capable of generating programs that can be ported into CICS/MVS environments.
- October 1988: A restructured IMS/ESA Version 3 with a separate Transaction Manager (TM) replaced IMS/VS with its separate Data Communications (DC) manager.
- February 1990: A statement of intent to increase interoperability between SAA and RISC/6000 AIX included detailed schedules on specific languages, services, and communications facilities.
- September 1990: System/390 (the 9000 series) was introduced, together with new MVS/ESA SP (MVS/SP Version 4), VM/ESA, and VSE/ESA operating systems; a new CICS/ESA; and new releases of TSO/E and IMS/ESA TM. The new names reflect a significant capability to exploit 9000 series features, whereas the new releases reflect fewer changes.

MVS

The MVS/ESA operating systems support three SAA environments. Although TSO/E, CICS/ESA, and IMS/ESA TM all have some capability to communicate across boundaries, each has its own work flow control that is logically independent of the others, governing its own languages, file structures, applications programs, and user interfaces. Both CICS and the Transaction Manager for IMS set up transaction processing environments, while TSO is a timesharing system more suited for program development. (See Table 2).

These environments are like operating systems loosely nested within an operating system. If MVS is running on a processor with a single instruction stream, it must, in fact, continuously interleave the execution of code from each of these environments as if they were multilevel application programs. (Multiprocessor systems with PR/SM partitioning can provide more integrity and better performance.)

Although they may all have some things in common, such as access to the DB2 database, each of the interfaces still has individual and unique characteristics in spite of SAA.

VSE

In the last few years, the DOS/VSE user base has successfully resisted IBM's attempts to "stabilize" the environment and channel upward growth into MVS. Although VSE is not a full SAA system, IBM has formally declared it the primary environment for distributed SAA transaction processing on small and medium systems. The VSE architecture was close enough to MVS and VM to support ongoing development from fallout programs during the lean years, and now the VSE user base is simply too large to be abandoned.

A CICS-oriented subset of SAA programming interfaces and communications services is supported under VSE. The communications interface defines an LU6.2-based protocol and related services for a consistent way of writing workstation-to-CICS programs, using the OS/2 EE as a front end. CICS programs can use the CPIC interface. This will allow creation of applications that conform to the Common User Access standards and that can be easily ported to other CICS environments.

Product Platforms

Supporting platforms are separated out into the primary product line(s) that the environment was designed for and other product lines with which it is upward and downward compatible. The 4381, originally just another member of the 4300 series, evolved into a separate series in its own right in the middle of the market as the introduction of the 9370 at the low end and 3080 at the high end.

The six fully implemented SAA software environments are associated with four hardware architectures.

PS/2: This workstation platform supports the OS/2 EE environment, a key environment for developing cooperative function-splitting applications using SAA. Although PS/2 is a workstation primarily for single-user client applications, cooperative front ends, or server functions on a LAN, it implements all SAA host elements and has the capability to operate as an APPN network node.

AS/400: This midrange platform supports the OS/400 environment.

System/390: The System/390 includes the 9000 and 3090/9000T series. The mainframe architecture introduced in September 1970 replaced all previous product lines (except low-end 9370 models) because of its 100-fold range of power.

System/370: The System/370 comprises the 3080, 3090, 4381, and 9370 series. Although System/390 is now the primary mainframe platform, SAA still supports older platforms loosely grouped as belonging to the System/370 architecture and its Extended Addressing and Enterprise System Architecture extensions.

Common User Access (CUA) Architecture

CUA defines a set of three types of user interfaces for SAA applications to be implemented regardless of differences in operating environment and hardware. It also gives guidelines on how to create and maintain the models it specifies. The intended result is that end users moving from one system to another will need little training as to how keyboards work or what screen formats represent as they encounter programs on different systems. The "look" and "feel" is the same.

The complete CUA specification does a lot more than provide guidelines on how the user terminal interface should look and feel, however. Manuals describe a framework for the development of consistent applications as well as consistent ways for applications to use windows, icons, direct screen manipulation, action bar choices, pull-down menus, dialog boxes, and color palettes.

The tools and interfaces that IBM supplies are only the beginning of developing and maintaining consistent CUA interfaces within an organization. Much of the CUA's characteristics will necessarily be tailored to the users' own environments. The keys to success, according to IBM, are establishing:

- Common Presentations: What users see should always be consistent as to visual appearance, and wherever possible, location.

Table 3. System Programs Used for CUA Development

Type and Level of CUA Development	Entry Model	Graphical Model	Graphical; Text Subset	Workplace Extension	On NPT	On PWS
CUA Capability						
NPT Terminal	•	—	•	—	•	—
PWS Workstation	•	•	—	•	—	•
Texts, menus, prompts	•	•	•	•	•	•
Full graphics, icons	—	•	—	•	—	•
Text, graphics for NPT	—	—	•	•	•	—
Complex procedures	—	—	—	•	—	•
System Program for CUA Development						
VM/CMS ISPF 3.2	•	—	•	—	•	—
CSP/AD 3.3	•	—	—	—	•	—
MVS TSO/E ISPF 3.2	•	—	•	—	•	—
CSP/AD 3.3	•	—	—	—	•	—
MVS IMS/VS MFS	•	—	—	—	•	—
MVS CICS/MVS BMS	•	—	—	—	•	—
CSP/AD 3.3	•	—	—	—	•	—
OS/400 DDS	•	—	—	—	•	—
OS/2 EE Dialog Manager	•	—	•	—	—	•
Presentation Manager	•	•	•	•	—	•
EASEL	•	•	•	—	—	•
CSP/AD 3.3	•	—	—	—	—	•

- Common Interactions: When interaction techniques associated with images and components are consistently supported, users develop an "ease of use" familiarity with them.
- Common Process Sequences: Maintaining consistent object-action interaction sequences trains users' expectations as to how the computer will respond. It is important that new applications remain compatible with the conceptual framework.
- Common Actions: Certain predefined actions have been given specific meanings by IBM. For example, selecting the "OK" bar tells the computer the user is done with a current task and wants to continue with the application.

Observations are also supplied as to the nature of the user interaction process and which interactions are best suited to CUA standards. For example, IBM finds it desirable not to focus on the mechanics of operations and observes that when users have to cancel what they are doing in order to do something else, they are "in a mode." Modes distract from the ability to conceptualize the application itself, just as complex interfaces do. IBM suggests providing good visual cues when it is impossible to avoid being in a mode, making the interface forgiving and responsive, and allowing the users to control the dialog as much as possible.

CUA Evolution

CUA is probably the only element of SAA that is universally recognized as a requirement for every single SAA application. It has undergone three levels of development.

1. 1987: The original guidelines provided models for basic nonprogrammable terminals (NPTs) as well as for programmable workstations (PWSs) that might or might not include graphics capabilities.
2. 1989: An expanded definition, which included icon-based processing and models for integrating CUA in the workplace, was introduced at the same time as OfficeVision.
3. 1990: A key role for the SAA PWS was clarified as that of a front end in cooperative processing applications; host-controlled NPTs were not going to support complex graphics capabilities. Programs providing support for program development of full graphics CUAs were going to be provided for OS/2 EE-based workstations, but not for host systems.

End-User Interface Definition

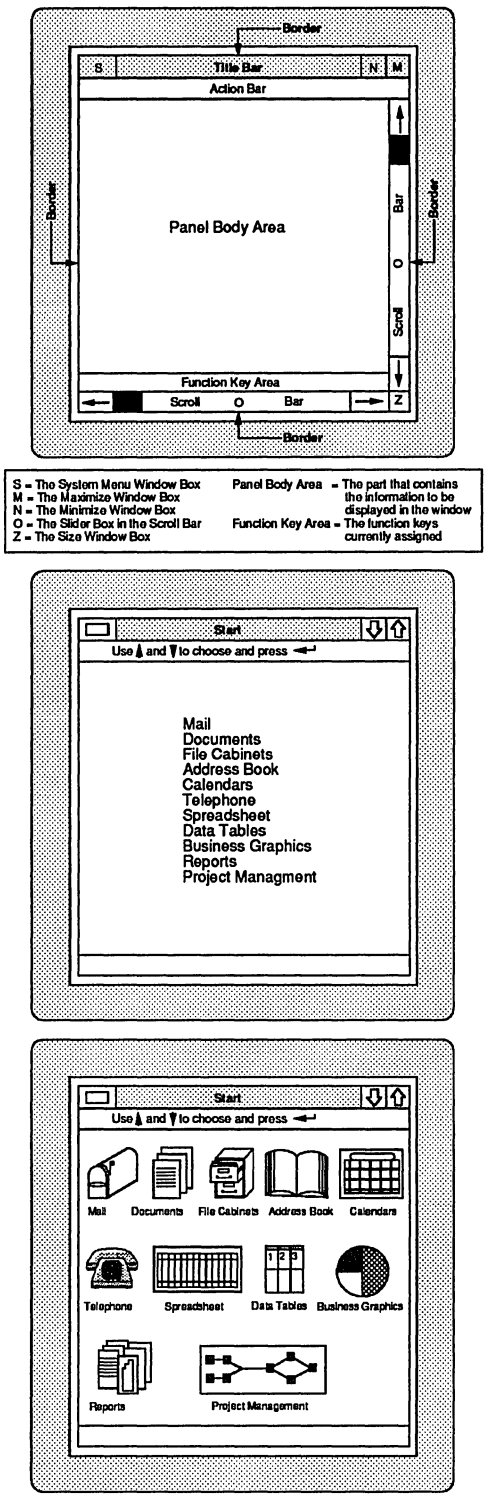
Specified screen layouts, menu presentations and selection techniques, keyboard layouts, and display options for SAA programmable workstations (PWSs) resemble those provided as standard features for OS/2 EE, while those for nonprogrammable terminals (NPTs) have evolved from 3270-type terminal keyboards. This means that in order to be SAA compliant, previously existing software including IBM's own system software will have to either convert the element of the program which sets up and controls user interfaces or supply two interface options, the old and the new. IBM has been changing the interfaces to its own system software for several years now.

The announcements in 1987 and 1989 defined three basic CUA architectural models, plus one subset, as follows.

Entry Model: Entry-level panels use the menus and prompts, familiar to keyboard operators working on data entry-intensive applications. This model, which is suitable for both NPTs and PWSs, is recommended for DOS-based PCs.

Graphical Model: Graphical-level panels make use of PWS windowing, graphics, and icon processing, so they are oriented toward OS/2 workstations. Also defined are action bars, pull-down menus, pop-up windows, checkboxes, and other graphical cues. DOS-based PCs can make limited use of this model if care is exercised not to impose on PC memory limitations.

Figure 2.
Basic CUA Panel Format with Two Implementations



The implementation at the center shows a basic alphanumeric panel, while the bottom one is a graphical panel with icons.

Graphical Model Text Subset: This interface is an NPT version that closely follows the graphical interface, although it cannot be precisely identical. Graphics may be limited or nonexistent, with textual phrases substituting for the diagrams and icons presented by the fully developed Graphical Model interface. It includes action bars, pull-down menus, and pop-up windows. Since processing is done on the host system, this model presents no difficulty for DOS-based PCs emulating NPTs.

Workplace Extension: Workplace extends the Graphical Model from a simple panel description to a model for integrating a variety of PWS applications. The relationship of the processes behind the icons for mail baskets, file cabinets, telephones, printers, and more is described. Each is treated as a separate service application (i.e., separate "object") that all end-user applications can invoke by means of the icon. Workplace is meant to be used with a mouse, so objects on the screen can be directly manipulated. OfficeVision/2 implements this interface.

CUA provides for both standardization and flexibility. As shown in Figure 2, the basic format can be manipulated to evoke the same interaction patterns between the user and the screen, despite the significant differences in the look of text-oriented and graphical screen designs. Screen designers can make use of windowing capabilities as well as graphics on both programmable and nonprogrammable systems, as shown in Figure 3.

Icons and Workplace Procedures

Icon input was originally introduced on personal computers like Apple's Macintosh so nontechnical users could cut down on the need for keyboard activation of simple localized functions. The user activates a cursor or mouse positioned over the icon that indicates a desired function. The procedure is easy to grasp intuitively. On the other hand, some users who are already adept with the use of keyboards complain that icons slow them down.

The use of an icon-oriented shell (or its equivalent) can invoke a considerable amount of behind-the-scenes activity that in itself invites the use of PWSs. If the workstation is tied into a distributed system and a function is activated, there is no need to know where the processing of the function occurred (unless the knowledge is desired).

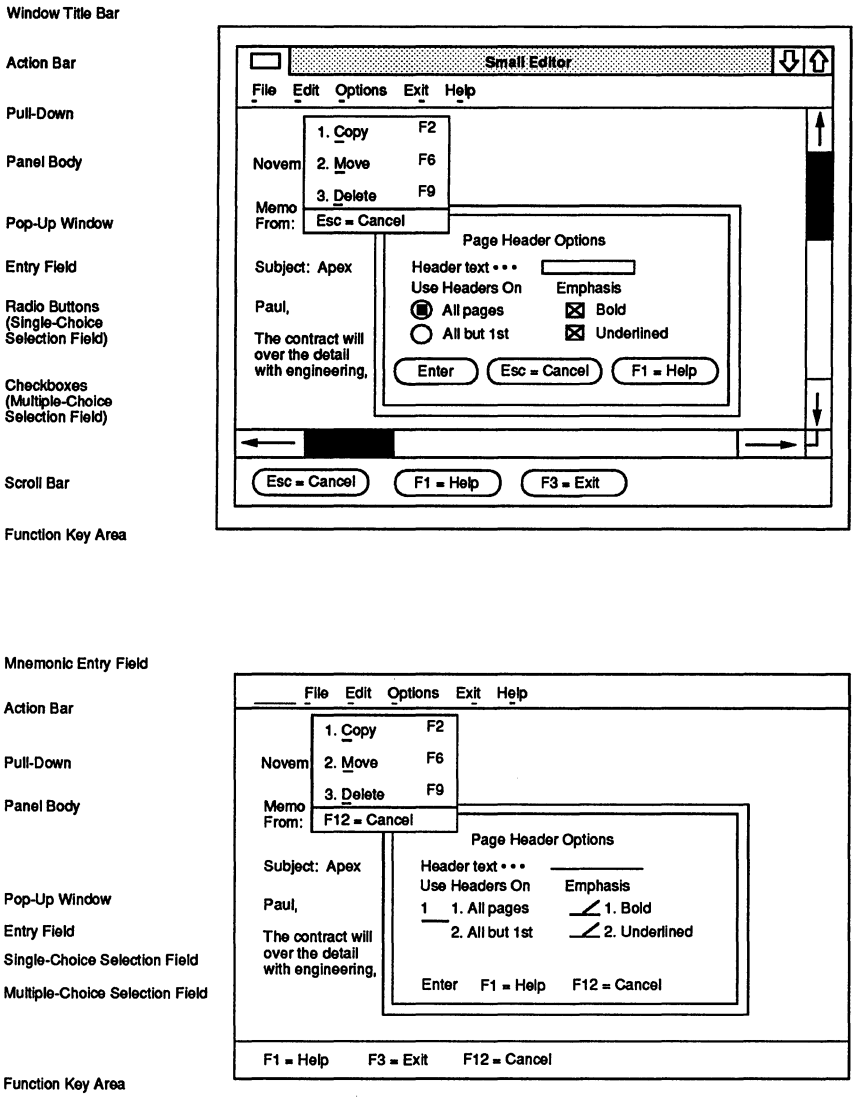
Generating and maintaining the icons themselves, processing the mouse input, and routing it to the proper subroutines are usually controlled by PWS code. There is no technical reason a traditional mainframe-controlled graphics terminal could not be coupled with mainframe- or mini-based icon processing.

However, it is less practical, not only because of the dropping price of general-purpose PWSs and the better response time achieved by having the graphics processing logic localized, but also because this configuration reduces the impact on host system performance caused by multiprogramming, contention for resources, and time-consuming I/O activity.

The use of the icon pictures themselves is not imperative to an icon-oriented system. Not all terminals need to use them. There is no reason why the word "printer" cannot be substituted for the iconographic picture of a printer, without disrupting any other procedure. The software that allows transparent, cooperative processing would work the same way, regardless of whether processes are activated by word symbols or picture symbols. It is the extra shell of

Figure 3.
CUA Panels with Windows

The top panel shows a typical CUA panel with windows for a programmable terminal, while the bottom panel is more typical of a nonprogrammable terminal.



code, separating the end user from the process, that really matters—not whether the shell generates words or pictures.

Tools for CUA Creation and Control

The primary tools used for creation and control of SAA CUA panels for NPTs and PWSs are the OS/2 EE Dialog Manager and Presentation Manager. Mainframes use the SAA-compatible Interactive System Productivity Facility (ISPF) program in MVS/TSO and VM/CMS environments and internal program modules in CICS and IMS/ESA TP environments for dialog management functions.

The OS/2 EE Dialog Manager on OS/2 EE-based systems and its mainframe counterparts are all interactive productivity tools for developing procedural programs that in turn create and control the CUA for a particular application. As Table 3 shows, these tools are suitable for developing Entry model CUA interfaces for NPTs and PWSs, or for Graphic subsets for NPTs, but are not suitable for developing full graphics and workplace-level interfaces.

Although the ubiquitous mainframe Graphical Data Display Manager (GDDM) program is capable of handling complex graphics and is accessible to all SAA environments, IBM has decided not to enhance its support program any further. Current industry trends toward cooperative processing and the off-loading of graphics processing have undoubtedly influenced the decision to restrict program development for these types of interfaces to the PWS on which they will eventually run. On OS/2 EE systems, IBM supplies not only the OS/2 EE Presentation Manager but also EASEL for support of program development. EASEL can be used to assist in the development of any type of CUA interface and was designed specifically to change existing applications with 3270 NPT interfaces into cooperative applications with PWS interfaces.

Furthermore, both EASEL and the Presentation Manager can handle event-driven graphics, not just procedural programming.

The Presentation Manager: The Presentation Manager is a CPI service program that provides for an event-driven object action process sequence, where an object is provided to be acted upon, rather than entering an action or mode and then determining which one of a universe of objects that action or mode will affect. The focus of attention is first upon any object that can be manipulated as a

single unit, and then next on the ways that users can modify, manipulate, or further create objects.

The Dialog Manager: The Dialog Manager is a productivity tool with a Dialog Tag Language (DTL) for programmers familiar with procedural techniques. It is not as well suited to developing graphical interfaces as the various types of SAA and non-SAA Presentation Managers, but it typically deals with checkboxes, list boxes, entry fields, and radio buttons in its panels. It supports three types of windows—primary windows, modal (pop-up) windows, and help windows—with the correct types of borders and components automatically supplied as circumstances dictate.

EASEL: EASEL is an application enabler that assists in developing cooperative front ends for existing programs that previously used 3270 NPTs, as well as for new programs. It is suitable for producing any of the CUA models, including Workplace. IBM has exclusive marketing rights to this program, which was developed in conjunction with American Software.

CUA Problem Areas

The first and foremost problem with CUA is that the user is the one who has to implement it. IBM can engineer its language compilers, services, communications subsystems, and so forth so that the user receives them unwittingly, but CUA has to be consciously engineered in keeping with SAA guidelines.

A second problem lies not so much in the interface itself, but in the timing of its introduction. If it had been here years ago, the weight of IBM's reputation in the market would have resulted in rapid acceptance of a de facto standard. Now there are a number of standards; IBM itself has maintained two major standards for screen and keyboard layouts for mainframe (3270 type) and midrange (5250 type) systems.

CUA Future Trends

Inevitably, CUA will evolve, but CUA standards will probably be more stable than CCS or CPI from the outset. IBM has stated that the company sees immediate requirements for the following:

- User ability to tailor the mouse interface, so that a tradeoff can be made between the simplicity of a single mouse button and the increased function available with multiple buttons.
- User ability to tailor the keyboard interface.
- Use of alternative input, such as touch.
- Use of context menus, which pop up next to icons.
- Augmentation of the action bar as screens increase in size and resolution.
- Additional techniques for dealing with database applications.

CPI: The Common Programming Interface

CPI currently specifies standards for the Application Generator (CSP/AE), C, Cobol, Fortran, PL/1, Procedures (REXX), and RPG language compilers or interpreters, as well as for interfaces to LU6.2-based CPI Communications (CPIC), Database (SQL and DRDA), Dialog, Distributed Data Management (DDM), Presentation, PrintManager, Query, Repository, and Resource Recovery services. This set of programs is the core of SAA, since they are the tools that are used to produce portable or distributable programs.

Standardization needs for the CPI include not only the elements of the compilers in and of themselves but also standardization of interlanguage communications and calls to supporting services. Supplying a comparable compiler for each environment is the first wave of compatibility (see Table 4).

Supplying comparable services in each environment is the second wave of compatibility (see Table 5). Creating a matching web of interlanguage calls is the third wave, the one that is least developed, as Table 6 shows.

The need to create a matching set of interlanguage calls on each system in order to remain compatible is what makes CPI like a single composite language interface, or a "shell." The SAA architecture emphasizes that programs should be modular and makes use of freestanding services with callable interfaces whenever possible.

When a program has imbedded calls to external facilities, making it portable means that there is a lot more to consider than just whether the source code written by the programmer will recompile on the new system. The external facility called upon must be there, too, and must respond in the same way when the recompiled calls are executed. For the same reasons, when IBM develops the capability to mix two languages in the same program, the same mix must be available for all SAA systems. At present, none of the current languages can call on every one of the services in every one of the environments.

CPI as a Key to Distributed Processing

Distributed processing is, in effect, a type of online portability. Programs or program segments must be capable of residing on more than one system. The difference is in the delivery system, which is usually a communications system that delivers updated changes to programs and data at regular intervals.

The trend toward distributed processing involves two elements: distribution of programs and distribution of data. In theory, the programs and data should be everywhere alike. In order to make this as close to reality as possible, a number of checks and balances need to be added to otherwise inviolate systems, and methods of efficient distribution, evaluation of possible communications overload, identification of possibly invalid versions of programs/data/transactions, and other unique procedures need to be developed.

CPI Language Compilers

In addition to the venerable Cobol and Fortran compilers, IBM included the young but fast-growing "C" compiler, the REXX procedures language, and the CSP Application Generator in the initial suite of SAA languages. C complemented the older compilers because of its all-around versatility, its capacity for bit manipulation, and its usefulness in developing system software. Despite the fact that REXX' previous history was limited to VM environments, IBM included it because it was so useful in its own niche. RPG, which had been previously used only at the midrange level, was added later in response to needs of

midrange systems to port their many RPG programs as they moved up. PL/1 was the last general-purpose compiler to be added.

C

"C" is a high-level language originally intended to simplify systems programming over multiple environments. As a result, it combines the ease of control, data structuring, and manipulation associated with high-level languages, with access to low-level, machine-related objects. The language is popular with engineers and scientists as well as systems programmers and is associated with the highly portable UNIX environments.

Standards for C continue to develop, but ANSI documentation is sufficient for a vendor to issue "standard versions" reasonably worthy of their names. The IBM versions of C closely follow the ANSI standard as described in the CBEMS X3J11 document dated April 1985. Since this standard differs in some respects from the versions implemented in most UNIX systems—and from the original description in "The C Programming Language" by B. Kernighan and D. Richie—IBM has also supplied a compiler option to select the desired set of conversions/coercions.

The Level 1 IBM mainframe C compilers that were introduced in August 1986 were SAA precursors classified as "Program Offerings," which is to say they were supported at a lower level than are IBM's "Program Products." SAA Level 2 C/370 compilers, introduced in September 1988, are upward compatible with Level 1. In February 1991, C/370 Version 2 added a global optimizer; features for user tailoring; and calls to CSP, QMF, and the ESSL libraries (in addition to those previously available). The VM/CMS compiler and the MVS compiler both produce the same object code, so each compiler can, in effect, be used as a cross-compiler for the other environment. C programs can dynamically link with supported MVS environments, such as IMS and TSO, and can directly invoke routines like GDDM. Since IBM itself is using C to write some of the SAA system software, the language is likely to be important for other SAA developers.

The OS/2 C/2 compiler, shipped first-quarter 1988, has been formally designated as an SAA compiler. C/2 has some searching/sorting and some string operations not supported on mainframe C.

Cobol

As today's primary high-level, business-oriented language, Cobol is a mature, well-known language. Even though it was first implemented in the early 1960s, its fairly English-like syntax has kept it popular during its evolution and expansion. The version of Cobol implemented for SAA systems is based largely on IBM's understanding of the ANSI Standard X3.23-1985 Intermediate Level, with four optional modules omitted (Report Writer, Communication, Debug, and Segmentation). Some elements of the ANSI 1985 High Level have been included, and IBM has added a few enhancements of its own, such as the COMP-4 and COMP-4 data items.

The SAA Cobol compilers and their related libraries are VS Cobol II Release 2 and later for MVS/ESA and VM/CMS (upward compatible from OS/VS Cobol), Cobol/400 for the OS/400, and Cobol/2 for OS/2 (upward compatible with PC Cobol Versions 1 and 2). Cobol/2 can run under PC-DOS 3.30 as well as under OS/2.

The Cobol compilers present more compatibility differences than do the other SAA general-purpose language processors. Although the mainframe VS Cobol II compiler runs under either MVS/ESA or VM/CMS, the two environments handle some operations differently, even though the OP code is the same. OS/2 Cobol 2 is based on the same standard but implements a greater number of features than the mainframe versions, even though its syntax for some operations is described by IBM as a subset of prior mainframe and PC versions. In addition, some differences are bound to occur in operations that are affected by machine-dependent characteristics, such as basic code type, internal representation of numeric operators, and others. For instance, native collating sequences, and any procedures which rely on them, use ASCII code sequences under OS/2 and EBCDIC under MVS and VM.

SAA manuals imply that many of the differences between Cobol/2 and VS Cobol II will disappear in the future; the trend will be to extend mainframe capabilities to match the greater flexibility and added features available in Cobol/2. For instance, every SAA Cobol program must have an Identification division; however, the Environment, Data, and Procedure divisions are all optional under OS/2, but required under MVS and VM.

One area of difference that illustrates the problems that can occur due to inherent differences in architecture, rather than the availability of specific instructions, concerns handling of numeric operands. Internal representations of numeric operands are system dependent by nature, since they involve approximations of decimal numbers by the computer's binary arithmetic facility, which varies in level of precision, etc., from system to system. Since intermediate results can thus be implementation specific, it is better to use *Add, Subtract, Multiply,* and *Divide* operators to achieve comparable results, rather than *Compute,* because the latter is a more complex statement and hence more likely to compromise program portability. Also, comparisons between numeric and nonnumeric operands may not be portable.

Despite these differences and, of course, depending on the nature of the application, the detailed documentation for each of the Cobol environments means that programmers can create highly portable applications. They must use care, however, to keep to the "lowest common denominator" of the instruction and, where system-dependent code must be used, to modularize the program so that the code is easy to convert.

CSP/AE Application Generators (but Not CSP/AD, EZ-PREP, or EZ-RUN)

The SAA Application Generator is based on the existing Cross System Product (CSP) sets, a product line that has developed over 30XX and 43XX mainframes, 8100s, and/or PCs for almost two decades. The announcement of IBM's AD/Cycle architecture in September 1989 increased CSP's importance.

Previously, it was not considered a crucial building block, due to its limited flexibility, slow development pattern, and only moderate popularity (4000 CSP/AD development systems installed in all possible environments). With AD/Cycle, however, the Application Generator became a key CPI component for joining together IBM and non-IBM programs that will control application development processes. As soon as IBM interfaces CSP to the repository product (which is planned for the near future), CSP will become a central alternative to third-generation languages like Cobol, Fortran, or C for SAA mainframe program development within the AD/Cycle architecture.

Table 4. SAA CPI High-Level Languages

Host Environment	MVS/ESA TSO/E	MVS/ESA CICS/ESA	MVS/ESA IMS/VS TM	VM/ CMS	OS/ 400	OS/2 EE	VSE/ CICS
C							
Latest Version	2.1.0	2.1.0	2.1.0	2.1.0	1.3.0	INA	2.1.0
Introduced	2/91	2/91	2/91	2/91	8/90	INA	2/91
Available	5/91	5/91	5/91	5/91	9/90	INA	5/91
Cobol							
Latest Version	3.2.0	3.2.0	3.2.0	3.2.0	1.3.0	1.2.0	3.2.0
Introduced	9/90	9/90	9/90	9/90	8/90	1988	9/90
Available	12/90	12/90	12/90	12/90	9/90	3/90	10/90
CSP/AE Generator							
Latest Version	3.3.0	3.3.0	3.3.0	3.3.0	1.3.0	1.3	3.3.0
Introduced	9/90	9/90	9/90	9/90	8/90	9/90	9/90
Available	9/90	12/90	12/90	12/90	9/90	12/90	10/90
Fortran							
Latest Version	2.5.0	—	—	2.5.0	1.3.0	1.2.0	—
Introduced	9/90	—	—	9/90	8/90	—	—
Available	12/90	—	—	3/91	9/90	3/90	—
PL/I							
Latest Version	3.1.0	2.3.0	3.1.0	3.1.0	1.3.0	1.3.0	1.6.0
Introduced	1990	9/90	9/90	9/90	8/90	10/90	9/90
Available	1990	12/90	12/90	12/90	9/90	3/91	10/90
REXX Procedures							
Latest Version	Level 2	—	—	Level 2	Level 2	1.3.0	—
Introduced	1990	—	—	1990	8/90	10/90	—
Available	1990	—	—	1990	9/90	3/91	—
RPG							
Latest Version	1.1.0	1.1.0	1.1.0	1.1.0	1.3.0	non-SAA	—
Introduced	2/91	2/91	2/91	2/91	8/90	—	—
Available	5/91	5/91	5/91	5/91	9/90	—	—

— =Not available.
3.1.0—Positional notation for Version 3 Release 1 Modification Level 0.

Acronyms:
CICS—Customer Information Control System.
Cobol—Common Business Oriented Language.
CSP/AE—Cross System Product/Application Execution.
CMS—Conversational Monitor System.
CPI—Common Programming Interface.
EE—Extended Edition.
ESA—Enterprise Systems Architecture.
Fortran—Formula Translation Language.
IMS/VS—Information Management System.
INA—Information not available.
MVS—Multiple Virtual Storage.
PL/I—Programming Language I.
REXX—Restructured Extended Executive.
RPG—Report Generator.
SAA—Systems Application Architecture.
TM—Transaction Manager.
TSO/E—Time Sharing Option/Extended.
VM—Virtual Machine.
VS—Virtual Storage.
VSE—Virtual Storage Extended.

These changes imply increases in IBM's CSP developmental activity that have already occurred and will continue to occur over the next few years.

CSP base programs combine productivity aids, a unique high-level language, and a cross-assembler to allow applications to be developed more rapidly or to be developed by less technically oriented programmers or end users. Critics claim that CSP sophistication has not quite reached the level where it could be called a "true" fourth-generation language, but that is clearly the direction of its development. CSP design is intended to cloak system-dependent characteristics, both in terms of the process of programming itself and the program created by the process. Program developers and users are buffered from the complexities of system and data management through the use of interactive, dialog-oriented procedures and syntax checking, fill-in-the-blank menus, prompting, tutorials, and a help facility.

CSP comes in two forms: an "AD" Application Development program and an "AE" Application Execution runtime system to provide what is needed to allow AD-developed programs to run. All CSP application development (AD) environments are designed to create one portable program that will run on a variety of application execution (AE) environments with a minimum of adjustments.

It is the CSP/AE environments which are the SAA standard, not CSP/AD. That is to say, CSP/AE versions will be provided for runtime control in all SAA environments, but not all SAA systems are guaranteed to support CSP/AD development systems. Obviously, the existence of CSP/AD is implied—or there would not be anything to run—but SAA itself does not architect this element. On the other hand, AD/Cycle does include CSP/AD.

The EZ-PREP and EZ-RUN programs for DOS and OS/2 environments have a number of differences from their CSP/AE equivalents on other systems. Among them

Table 5. SAA CPI Service Interfaces

Host Environment	MVS/ESA TSO/E	MVS/ESA CICS/ESA	MVS/ESA IMS/VS TM	VM/ESA CMS	OS/400	OS/2 EE	VSE/CICS
CPI-C Communications							
Earliest Version	2.3.0	3.2.0	3.2.0	1.1.0	—	—	—
Introduced	9/90	9/90	9/90	9/90	SOD	SOD	—
Available	3/91	6/91	6/91	12/90	—	—	—
Alternatives	LU6.2	LU6.2	LU6.2	LU6.2	LU6.2	LU6.2	LU6.2
Database-SQL							
SQL Program	DB2	DB2	DB2	SQL/DS	SQL/400	DB Mgr.	SQL/DS
Latest Version	3.3.0	3.3.0	3.3.0	3.3.0	1.3.0	1.3.0	3.3.0
DRDA Version	3.3.0	3.3.0	3.3.0	3.3.0	SOD	1.3.0	—
Introduced	9/90	9/90	9/90	9/90	8/90	10/90	9/90
Available	10/90	10/90	10/90	6/91	9/90	3/91	10/90
Dialog Manager							
Primary Program	ISPF	BMS	TM MFS	ISPF	—	D. Mgr.	BMS
Latest Version	3.2.0	3.2.0	3.2.0	3.2.0	—	1.3.0	3.2.0
Introduced	9/90	9/90	9/90	9/90	SOD	10/90	9/90
Available	12/90	12/90	12/90	12/90	—	3/91	10/90
Alternatives	—	—	ISPF	—	—	—	—
Presentation Manager							
Primary Program	—	—	—	—	—	P. Mgr.	—
Latest Version	—	—	—	—	—	1.3.0	—
Introduced	—	—	—	—	—	10/90	—
Available	—	—	—	—	—	3/91	—
Alternatives	GDDM	GDDM	GDDM	GDDM	P. Mgr.	EASEL	GDDM
PrintManager							
Latest Version	1.1.0	—	—	1.1.0	1.3.0	—	—
Introduced	9/90	—	—	9/90	8/90	—	—
Available	6/91	—	—	6/91	9/90	—	—
Query Manager							
Primary Program	QMF	QMF	—	QMF	Q. Mgr.	Q. Mgr.	—
Latest Version	3.1.0	3.1.0	—	3.1.0	1.3.0	1.3.0	—
Introduced	9/90	9/90	—	9/90	8/90	10/90	—
Available	12/90	12/90	—	12/90	9/90	3/91	—
Repository Manager							
Latest Version	1.2.0	—	Via TSO	—	—	1.2.0	—
Introduced	9/90	—	9/90	—	—	9/90	—
Available	3/91	—	3/91	—	—	3/91	—
Resource Recovery							
Primary Program	—	—	IMS TM	SQL/DS	—	—	—
Latest Version	—	—	3.2.0	3.3.0	—	—	—
Introduced	—	—	9/90	9/90	—	—	—
Available	—	—	6/91	6/91	—	—	—

— =Not applicable.
3.1.0—Positional notation for Version 3 Release 1 Modification Level 0.

Acronyms:
BMS—Basic Mapping Support.
CICS—Customer Information Control System.
CMS—Conversational Monitor System.
CPI—Common Programming Interface.
CPI-C—Common Programming Interface for Communications.
D. Mgr.—Dialog Manager.
DB Mgr.—Database Manager.
DB2—Database2.
DRDA—Distributed Relational Data Access.
EE—Extended Edition.
ESA—Enterprise Systems Architecture.
GDDM—Graphical Data Display Manager.
IMS—Information Management System.
LU6.2—Logical Unit 6.2 protocol (base of CPI-C).
MVS—Multiple Virtual Storage.
P. Mgr.—Presentation Manager.
Q. Mgr.—Query Manager.
QMF—Query Management Facility.
SAA—Systems Application Architecture.
SOD—Statement of Direction for Future Support.
SQL/DS—Structured Query Language/Data System.
TM—Transaction Manager.
TSO/E—Time Sharing Option/Extended.
VM—Virtual Machine.
VS—Virtual Storage.
VSE—Virtual Storage Extended.

are differences in the way SQL-related calls, file changes, and file recovery are handled. Other differences involve calls to non-CSP programs, limitations in map variable field edit routines, and reversed order of storage of binary fields (yielding different results on the PC than in other environments).

The introduction of OS/400 Version 1 Release 2, which predated AD/Cycle by a few months, had also provided for an integrated CSP/AE module capable of running programs developed by a mainframe CSP/AD Version 2 Release 2 Modification Level 1 or later program. This is the first time CSP has generated programs for midrange systems based on S/3X architectures.

The AD/Cycle standard extended architected CSP support to include the CSP/AD 3.3.0 or later application generators. The needs of AD/Cycle dictate that the CSP product set must evolve into true 4GLs that can generate running code from a minimum of information, supplied in English-like interactive conversations.

CSP/AD Version 3 Release 3, released in the same month as AD/Cycle, includes two important features—an external source format (ESF) facility first introduced at the 3.2.0 level and a programmable workstation (PWS) access facility. Both are required if the application generator is to be used with AD/Cycle.

CSP/ESF provides a standardized way of handling external source formats (ESFs), which allow interfacing of CASE tools, importing of application tools from other vendors, external object manipulation, and better library and configuration management. However, CSP still lags behind other CPI components in the accommodations provided for interfacing with other SAA languages and services.

The CSP/PWS workstation feature (ship date June 1990) is particularly significant, since it allows CSP itself to run as a cooperative processing system, in a limited way. Previously, only interactions with nonprogrammable terminals (NPTs) or workstations that emulated them were accommodated.

Fortran

Fortran is to the scientific and technical user what Cobol is to the business user—a language that is equally venerable and important for its sphere of operations. This language was originally developed by IBM. The current SAA compilers conform to the specifications of the ANSI X3.9-1978 (Fortran '77) standard and the ISO 1539-1980 specification but include some enhancements to these standards. For instance, all the IBM SAA versions are capable of using names up to 31 characters long. The SAA mainframe compiler is VS Fortran Version 2 (5668-806), which runs under both MVS/ESA and VM/CMS.

Compared with the number of problems a Cobol programmer faces, at least initially, there are relatively few differences between OS/2 and mainframe Fortran implementations to cause SAA compatibility problems. The most noticeable difference is that the *Include* statement, an extension available to Cobol/2 users, is not implemented for mainframe environments. For VS Fortran Version 2, users are advised to use the *OCSTATUS* execution option, while OS/2 users are advised not to use the /I compiler option, in order to maintain SAA conformance.

Since IBM frames are coded in EBCDIC, whereas PCs use ASCII, difficulties arise with any instruction that deals directly with system codes. For example, external files with system-dependent names can be referred to by some instructions. Unformatted records with values in system-dependent forms are not converted to internal forms when read or written, but formatted records coded in EBCDIC or ASCII (as appropriate) are converted to internal values when written or read. The basic system code also affects collating sequences.

One highly significant set of differences to Fortran programmers lies in the range of values and level of precision possible for approximations of real numbers, due to the different ways real (floating point) arithmetic can be implemented. For example, the range of values for mainframe VS Fortran II includes 10 (-78) to 10 (+75), 0, and -10 (-78) to -10 (+75) (bracketed numbers represent exponents). These limits remain the same for single-precision (four bytes) and double-precision (eight bytes) numbers. The Fortran/2 range of values varies depending on whether single- or double-precision types are used and whether the value is normalized or denormalized. Normalized single-precision numbers include 1.2 x 10 (-38) to 3.4 x 10 (+38), 0, and -1.2 x 10 (-38) to -3.4 x 10 (+38), whereas normalized double-precision numbers include 2.23 x 10 (-308) to 1.79 x 10 (+308), 0, and -2.23 x 10 (-308) to -2.23 x 10 (+308). The OS/2 single-precision range is half that of the mainframe, but the double-precision range is four times greater. Fortran/2, moreover, provides positive infinity, negative infinity, and NaN (not a number), data types not supported under VS Fortran II.

The differences in the two compilers are interesting since one would expect the mainframe compiler to be the more flexible and the more powerful. Of course, mainframes used for scientific and engineering work can access special vector-processing options and extended Fortran capabilities, but these facilities are not found in SAA. Users who employ OS/2 systems for Fortran program development should avoid double-precision arithmetic if the program is to run on a mainframe.

IBM has dutifully released a Fortran compiler for the AS/400 series, in order to keep the SAA "on all systems" promise, but it is unlikely to be widely used because the internal architecture of the AS/400 does not lend itself to technical applications. IBM has, in fact, said that this compiler will not be developed further.

PL/1

Although Programming Language/1 (PL/1) has been used at all levels of the market for many years, it was not added to SAA until 1989. It was designed as an attempt to create a programming language that would combine the virtues of both Cobol and Fortran without their vices, but it never succeeded in supplanting the two older languages. The smaller overall user base may be why IBM did not initially include it in SAA. Mainframe PL/1 supports calls to C, Cobol, GDDM, ISPF, REXX, and SQL. OS/400 PL/1 supports calls to SQL.

Procedures Language: REXX

The Procedures Language for organizing system development and run-time procedures is based on the "REXX" System Product Interpreter that had been an inherent part of the VM operating system. It was originally designed for character string manipulation. In SAA, it operates as an application macro processor as well as a system procedure language for developing portable procedures, even though SAA systems are not inherently compatible.

Until after the introduction of SAA, this language had not been offered as a separate, freestanding entity in environments outside of VM, but IBM itself has used it within the organization in conjunction with the VM operating system and the internal IBM VNET network. Now REXX is available in all SAA environments except CICS and IMS TM and has been broken out as a freestanding compiler under VM.

REXX is a clear, easy-to-learn language that makes it amenable not only as a macro and command language but also as a full-function development language. It includes structured programming constructs, external and internal calling mechanisms, a wide variety of expressions and built-in functions, extensive string parsing by pattern matching, exception handling and tracing mechanisms, presentation of host commands to the system, and many more features.

REXX was added as an integral part of TSO/E in 1988 and can be used by NetView together with or in place of CLISTS. It was added to the OS/400 operating system in August 1990. It has not been included as an element of CICS or IMS environments. The ship date for the OS/2 EE was March 1990, since the procedures language was included as part of OS/2 EE R.1.2. The OS/2 EE version can call C, the Query Manager, and SQL.

RPG General-Purpose Language Compiler

The RPG Report Generator, as its name implies, started out as an easy-to-use way of generating reports for users of midrange systems like System/36 and System/38. An entry-level user could specify input and output formats through fill-in-the-blank menus, and the highly structured compiler turned this data into the necessary code for processing input and generating output.

From these beginnings, RPG has evolved into a full-function, nonprocedural general-purpose language. Programmers who want to create complex batch or interactive programs can structure programs using a full set of file control operations and facilities like DO, IF, and ELSE.

The SAA RPG specification is based on the RPG/400 compiler, which was immediately available for AS/400 systems at first delivery in August 1988. The WORKSTN device type is not a part of SAA RPG but can be used for compatibility with RPG III. This version can call C, the OS/400 Presentation Manager (which is partially SAA compliant), and SQL.

PS/2 systems do not have a compatible SAA RPG compiler but have an interpreter that allows them to run RPG programs developed on another system. The big news for RPG occurred in February 1991, when an RPG/370 compiler was released for MVS/XAMVS/ESA and VM/SPVM/ESA environments. The lack of this compiler made it unlikely that midrange system users would move applications up to mainframes, because RPG is more prevalent than Cobol at the midrange level. This compiler, which requires the C/370 library for both program development and execution, can run in conjunction with SQL/DS or IMS database managers, ISPF dialog manager for MVS/TSO or VM, and the REXX procedures language. In MVS environments, it operates on TSO/E and IMS/VS DC or IMS/ESA TM subsystems, but not CICS.

CPI Services

CPIC Communications Interface and LU6.2 APPC

Program-to-program communications, in which two programs interact without having to specify details of the underlying communications vehicles, is the core technology for SAA. As of September 1991, most of the Advanced Program-To-Program Communications (APPC) in SAA environments that support this type of interaction supply not one but two possible ways to invoke this capability.

First, there is the SAA standard, the CPIC Communications Interface, which involves instructions and data formats that are standardized for every environment (although not for every language). The same CPIC is available on OS/2 EE as on MVS. CPIC in the SAA CPI level is processed by LU6.2 code in the SAA CCS level, but it presents an easier to use interface than the LU6.2 interface. Second, there is the more modular LU6.2 standard which has been endorsed as part of the CCS communications subsystem. However, since it is modular, it can vary considerably from system to system. There are a number of LU6.2 applications programming interfaces (APIs) that differ because of the modularity of the underlying support subsystem and because they are essentially a lower level interface than CPIC. SAA users could use them, but doing so might compromise program portability if the usage were not carefully controlled to stay within the bounds of the facilities held in common with CPIC.

SAA applications can function without CPIC and/or LU6.2, but handling cooperative or distributed applications without them is not IBM's intention for SAA.

Database Interfaces (DBIs)

The Structured Query Language (SQL) residing in DB2, SQL/DS, SQL/400, and OS/2 EE is a nonprocedural, English-like language used to address IBM's relational database products, in either an interactive mode or through program calls. The Distributed Relational Data Access (DRDA) interface, which is part of some versions of SQL, provides additional instructions needed for access and control of distributed database versions. SQL (and its corollary standards) is one of the truly core components of SAA (the other is CPIC). Nearly every one of the CPI languages and services has precompilers or other facilities for calling or integrating SQL code.

Structured Query Language (SQL): SQL is a language that was developed in IBM laboratories together with the relational model. It has proven to be one of IBM's real successes in cross-environmental implementations. Furthermore, SQL is widely accepted as a standard by third-party developers and is the basis for standards endorsed by both ANSI and the ISO standards organizations.

The implementing products for the SAA SQL Database Interface are as follows.

- Database 2 (DB2)—Version 1 Release 3, Version 2 and up for MVS (TSO/E, CICS, and IMS environments).
- SQL/Data System (SQL/DS)—Version 2.1 and up for VM and VSE environment.
- SQL/400—A separate SAA-compatible interface to OS/400, which has an imbedded nonstandard relational database manager.
- OS/2 Database Manager—Integral to OS/2 Extended Edition Version 1.1 and up.

As previously noted, a high degree of standardization does not mean that a language operates identically in every environment. With SQL, storage structures, such as spaces for tables and indexes, differ in each implementing product; so do the maximum precision of a decimal number and the floating-point approximations of real numbers. Some of the other environment-dependent differences include the syntax of graphic string constants and the comparison of varying length strings.

OS/2, for example, does not support the SQL authorization scheme, graphic strings in the single-byte version, and Date/Time arithmetic operands in SQL or the UNION operator. In the VM environment, *Describe* and *Using Descriptor* statements are not supported in Cobol or Fortran environments, and the imbedded *Select* (or the INTO clause of the SELECT statement) cannot be used with Fortran. In most cases a host variable can be up to 18 characters long, but the limit is 6 characters for Fortran in a VM environment.

For the most part, however, correlation between the environments is high, both in terms of the specific operations supported and in the way operations behave. The advent of the S/390 9000 series was accompanied by a new version of DB2 that included a new, open, high-level language interface. This allows not only C, Cobol, Fortran, and PL/1 to directly interface to DB2 but also the new MVS/APPC, which includes the CPIC SAA communications interface. Previously, an assembler code module was required between these high-level language calls and the call interface. The QMF SAA query language, SAA CSP application generator, and ISPF dialog manager can also interact with DB2.

DB2 Version 2 Release 3 complies with FIPS 127.1, ANSI X3.135 1989, ANSI X3.168 1989(E), and ISO 9075 1989 standards. In VM systems, SQL interfaces with C, Cobol, CPIC/APPC, CSP, ISPF, PL/1, RPG/370, and QMF. The SQL/DS database Version 3 Release 2 complies with FIPS 127.1 standards, as well as ANSI X3.135 1989, ANSI X3.168 1989, and ISO 9075 1989.

SQL/400 interfaces with Cobol, CSP, and RPG languages, while the SQL facilities in OS/2 EE R.1.2 and later interface with C, the EZ-RUN version of CSP, Query, Cobol, and REXX. A Statement of Direction was issued February 15, 1990 to provide AIX systems with databases which will conform to SQL standards and will interoperate with SAA distributed database capabilities.

CPIDBI Distributed Relational Database Architecture (DRDA): DRDA defines an aspect of an SAA relational database architecture that is accessed and manipulated through instructions which are extensions of SQL. These include a connect statement and coded character set conversions, several language extensions, referential integrity, and common SQL return codes. Distribution of data and related types of processing involve a number of permutations and various levels of access capability. In addition to new elements provided in SQL, the SAA QMF query facility has new distributed retrieval functions allowing transparent access to distributed relational data.

The procedures currently available allow the user to:

- Retrieve data from another system and work on it locally
- Work on data on a remote system and retrieve the result
- Swap relational data among systems in order to keep all up to date

The individuality of the terminal-to-computer interaction and the standards already developed (LU6.2, etc.) have reduced the architectural problems for implementation of the first two interactions. However, swapping relational data among previously incompatible databases has required much development work.

Up until September 1990, distributions were only possible between selected SAA databases of the same type, but on that date, the first exchanges between nonidentical databases were announced. (See Table 7.) This capability is developing slowly at the low end: OS/2 EE database facilities cannot swap relational data with DB2 or SQL/DS databases, and nobody swaps anything with the AS/400 yet.

One of the justifications for data distribution is to send it close to the source of processing when the source is far from the centralized point of overall control. If the source is nearby, why not just have users access the central database? Thus, data distribution implies networking, especially to very remote locations. The best of all possible worlds for very remote locations would be a situation where data could be rapidly handed from network to network, regardless of network type, until the data reached its final destination, rather than requiring a far-flung system to run a long, expensive direct-connect line into a central network.

AS/400 APPN's dynamic addressing features are much better adapted to handling the type of communications underpinnings needed by distributed databases than the current level of mainframe SNA. With APPN, if a database cannot be located on the local network, it is even possible to broadcast the needed address to an unknown destination on another attached APPN network in an attempt to find it. Although APPN and SNA can interconnect, doing so limits each of them, in terms of extensions that users are usually unhappy to give up.

The end result is that problems have been resolved to the point of allowing flexible dynamic exchanges only within SNA subareas, or if the mainframe is treated as a Low Entry Networking (LEN) node within APPN. Yet, to automatically send relational files back and forth between the AS/400 and DB2, or SQL/DS, or OS/2 EE suggests that passage through multiple SNA and APPN networks would become a commonplace need, rather than the rarity it is now. At the present level of network development, APPN or SNA can handle data distribution passing within and through its own type of network, transparently. It is possible to hop to each other's network, but hopping through each other's networks is definitely not a transparent operation, especially several in succession. It is likely that even when the distribution of data is architected for all database exchanges, true transparent distributions across multiple network types will not be immediately forthcoming.

Dialog Interfaces (DIs) to Dialog Manager, ISPF, and EZ-VU

SAA dialog services are enablers for the development of interactive applications, especially those that will be using the Entry or Graphical Subset types of CUA user interfaces. The dialog panels control user interactions by means of menu selections, help information, data requests, and messages. A dialog tag language (DTL) allows the programmer to define dialog objects (panels, help panels, messages, keylists, commands) which are independent of the operating environment.

The SAA Dialog Interface is based on the PC EZVU II facility, which is, in turn, an outgrowth of the mainframe Interactive System Productivity Facility (ISPF) programs. ISPF is an important productivity aid for developing interactive applications; assisting in the development of panels, menus, help, data requests, system messages, etc.; and in buffering against many device-dependent considerations. Most VM and TSO/E environments use ISPF, but it is not yet supported in CICS or IMS/VS environments.

ISPF is functionally equivalent to a subset of OS/2 EE Dialog Manager, but the syntax of the corresponding calls

is different, so it is not being labeled a full SAA facility. ISPF Version 3 Release 1.0 added CUA capabilities to earlier releases; Version 3.2.0 added "Workstation Platform," a feature that can assist in developing SAA cooperative processing. This feature downloads code extensions from ISPF's Program Development Facility (PDF) Version 3 Release 2 to allow OS/2 EE 1.2 workstations to "check out" (receive) objects controlled by a host's ISPF Software Configuration and Library Manager (SCLM). SCLM library function (and control) is extended directly to the workstation.

The result is flexibility. Programmers or users can utilize their workstation to view and operate on objects while they reside in the host, as was done before, or make the same type of interchanges locally in the workstation, with the option of uploading the result. Data communications for file download/upload is controlled by the communications manager that is an integral part of OS/2 EE 1.2.

However, the change in emphasis toward having the PWS itself be the core for development of cooperative processing (especially graphics) front ends has downplayed the importance of this feature. ISPF 3.3.0 continued support but did not enhance the facility.

The SAA Dialog Interface for the AS/400 series is a separate facility that extends the capabilities of the native dialog manager. The new combination was released in August 1990. The ship date for the OS/2 EE Dialog Manager was March 1990, as part of OS/2 EE 1.2.0.

Distributed Data Management (DDM)

Interface Distributed Data Management (DDM) is an evolving SAA architecture which allows systems of different types to exchange records, files, or relational data. It specifically defines system-independent structures, command formats, parameters, objects, and messages so that applications programs can access files from different types of systems. The DDM Level 1.0 implementation enables different systems to exchange record-oriented files; DDM Level 2.0, introduced in June 1988, adds exchanges of stream files (documents) and directories (folders). DDM Level 3.0, introduced in September 1990, enables exchange of relational data in conjunction with DRDA interfaces. To some extent, systems can also perform remote operations on each other's files without an exchange.

DDM uses three programs to add another processing layer to the basic file request procedures on both the requesting and the serving system. In addition to a communications server on all systems, an SDDM program on the source/requestor/client system and a TDDM program on the target/server system work together like elaborate protocol converters. (See Figure 4.) These modules establish conversations with each other by means of IBM's LU6.2 Advanced Program-to-Program Communications (APPC) standard, using SNA, LEN, APPN, or X.25 networks as transport mechanisms.

The basic procedure is as follows. When an application wants a file, local system software first verifies that it does not have it in the usual local directory, and then the request is trapped and sent to a supplemental DDM side directory, which must be maintained by the system administrator. The SDDM translates the request into standardized DDM commands to ask for the desired file and uses its own command set and communications modules and the system's APPC facilities to establish a session between itself and the TDDM program on the system that has the desired file.

The targeted system's TDDM server program receives the standardized DDM code and translates it into commands specific to its own system in order to retrieve the files. After obtaining the file, the TDDM reverses the translation process, changing local commands and structures back into standardized DDM commands and structures so that a copy of the file can be sent to the requesting SDDM. The requesting system, when it receives the code, again has to go up the translation ladder to present it in an acceptable form to the original requesting application.

The theoretical goal of DDM is that an applications programmer can ask for a file without knowing its location or its characteristics. DDM does achieve this to a high degree, but differences in the way instructions, files, and directories are handled in various native architectures now make it difficult to achieve fully.

The primary arena for DDM activity has been the midrange, where systems can function as both Target DDM file servers (TDDMs) for each other and for PCs, or as Source DDM (SDDM) requestors from each other and from mainframes. Mainframes running CICS (MVS or VSE) environments can only support a TDDM server program, and mainframes cannot be requestors. PCs, on the other hand, can only be DDM requestors.

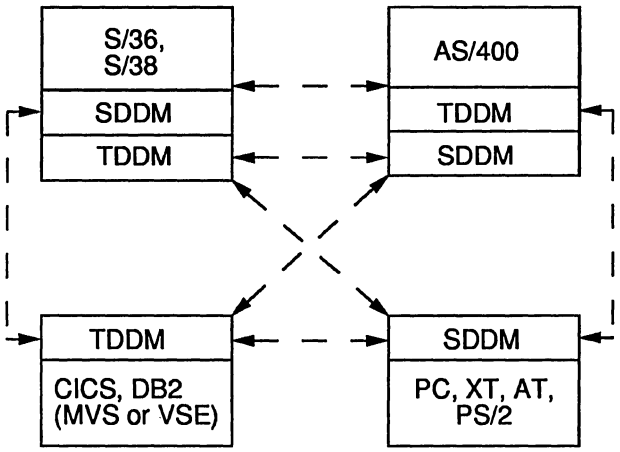
It is not yet clear whether designating DDM as an SAA element means that all SAA environments will eventually operate as both requestors and servers, since the software for these two functions is always implemented in separate SDDM and TDDM modules. Adding relational data exchanges seems to indicate that this may be the case, since exchanges will be needed in many distributed systems.

Although this double-conversion procedure would seem to provide an opportunity for completely transparent data transfers, there are in fact many problem areas. To give some examples, Delete instructions may work in one context but not in others. If one of the various types of VSAM data sets is not allowed to be deleted in its native CICS architecture, but its equivalents elsewhere can be, a programmer must remember that a Delete instruction that works elsewhere will not work when sent to this file on a CICS system. Keyed files containing zoned, packed, or other than character fields can return unexpected results when created by an AS/400 on a System/36 or System/38 because the latter systems do not support these types of fields. System/36 allows copying a remote file to another remote file, but the AS/400 allows only one of the files to be remote.

There are other ways to transfer files. DDM is oriented toward interactive processing at the record level, although it is fully capable of large file transfer operations as well. When transferring all the records in a file, where there is to be no application processing of the data, alternatives include user-written APPC programs, system-specific object distribution programs, and SNADS. APPC programs (of which SNADS and DDM are examples) can operate over APPN networks as well as over traditional mainframe SNA network backbones. IBM states that in general, file transmission times using DDM, SNADS, and user-written APPC programs are usually within 10 percent of each other.

Use of the SNDNETF (Send File) CL command via SNADS offers the advantage of transmitting one copy of the data to more than one system through a multiple-node network. Distribution can also be time-scheduled using the SNADS distribution queue parameter. SNADS transmissions require a copy of SNADS on all source, target, and

Figure 4.
DDM Clients (SDDMs) and Servers (TDDMs)



intermediate nodes. Target systems require processing by a RCVNETF Receive File, a step that would not be required by DDM.

DDM file conversion and file operations occur at both ends of the DDM connection, where most system-dependent and operation-dependent variables reside, but in general, performance impact comes predominantly from the communications line usage.

Presentation Interfaces (PIs) to Presentation Manager and GDDM

Many end users feel a standardized presentation interface for control of graphics is probably second in importance only to the overall end-user access interface standard. This is logical, since a Presentation Manager is needed to code the more complex types of graphics and workplace CUA interfaces.

Although the SAA Presentation Interface is based largely on the mature Graphical Data Display Management (GDDM) set of System/370/390 programs, IBM has decided to restrict SAA endorsement of the CUA program development facilities to the OS/2 EE Presentation Manager. This decision was ostensibly based on the idea that the Presentation Manager is used to develop the type of interface that runs on programmable workstations (PWSs), not on nonprogrammable terminals (NPTs). It is more logical to use a cooperative PWS to develop a PWS interface than it was to use an NPT and a host and then download the interface into a PWS. That being so, GDDM might be used to support NPT graphics, but there was no need to endorse GDDM for CUA development.

The OS/2 EE presentation interface merges the equivalent of GDDM with a windowing capability based on Microsoft's Windows. The GDDM presentation standard specifies a set of functions that allows display and printing of alphanumeric data, vector graphics, raster graphics, and image data on a variety of devices, including plotters, personal computers, color displays and printers, monochrome displays and printers, and so on. Major functions include comprehensive graphics support, limited image support, saving and restoring of graphics pictures, and implementation of keyboards and mouse devices in such a way that applications can also conform to the Common User Access standards.

The main GDDM subroutines can be directly called as part of major programs in many environments. Examples are OS/370 Assembler, Cobol, PL/1, VS APL Release 4, and, in some cases, Fortran and Basic; IMS, CICS, TSO, and CMS each support direct access to GDDM by some of these languages and not others. GDDM's applications programming interface (API) is consistent across displays, printers, scanners, and graphics devices. Datastreams supported include the 3270 Extended Data Stream, the SCS SNA Character String, and the IPDS Intelligent Printer Data Streams, all of which are specified by the SAA architecture.

The GDDM-Restructured Extended Executor (GDDM-REXX) and GDDM Graphical Kernel System (GDDM-GKS) are particularly noteworthy in terms of the interlocking standardization efforts of the SAA architecture. GDDM-GKS supports ISO and ANSI international standards for graphics file definition (as described in ISO 7942-1985[E], ANSI X3.124-1985, and ANSI X3.124.1-1985 documents) as an adjunct to the base GDDM program. GDDM-REXX provides a direct interface between the base GDDM/VM program and the REXX/VM Interpreter facility of VM/CMS CMS, the basis of the SAA procedures language. GDDM-REXX allows systems programmers to add GDDM calls to REXX EXECS, including calls to the GDDM base API, GDDM-PGF API, and GDDM-GKS API, but not to the GDDM/graPHIGS API. It is thus one of the important bridges linking SAA parts.

The complex of interlocking programs can provide a wide variety of capabilities, but the inevitable question is, at what cost? One of the important rationales for adding PWS is to download graphics processing from a high price-per-MIPS system like a mainframe to a low price-per-MIPS system like a PC or PS/2. This trend is undoubtedly another reason why IBM does not want to promote GDDM as a candidate for full-scale SAA development.

PrintManager Interface

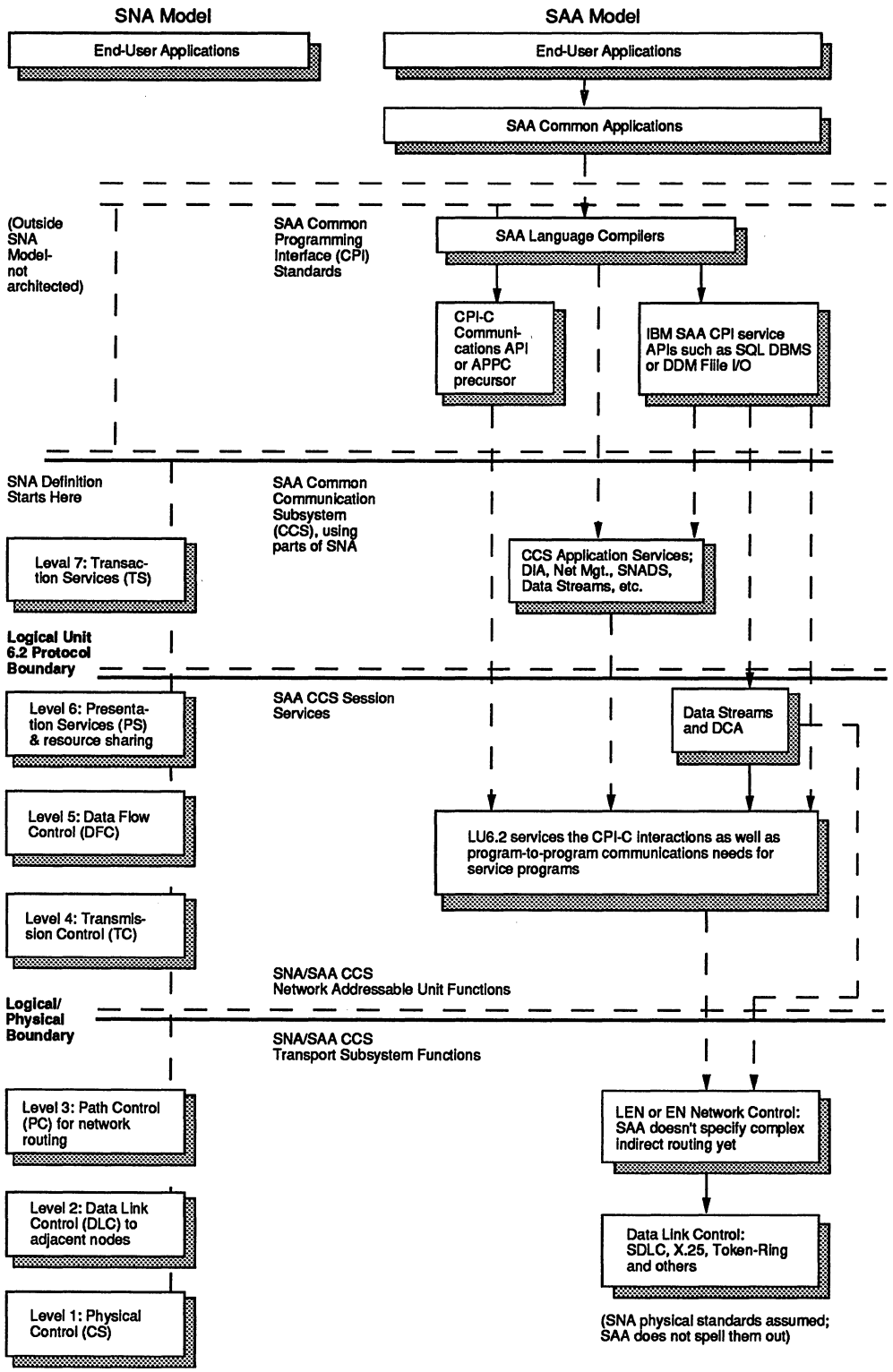
PrintManager services provide a uniform software interface for a variety of types of printers, in order to standardize printer interactions and operations. Standards are established for print instructions, formats, fonts, and so forth, so that the programs are dealing with logically identical printers rather than the individual characteristics.

PrintManager goals are closely related to SystemView's integrated system management strategies and the CPI Resource Recovery Interface (RRI). The overall goal is to have automatic detection of a failed resource and provision of an alternative, so that when a user tries to use it, the system software automatically routes requests to the alternative device. Then, when the device is restored, user requests automatically revert to their originally intended destination.

The roles played by these three components are:

- SystemView describes the overall operation and how the pieces fit together with other system operations.
- RRI Services manage the pool of shared devices which needs to be established to be able to automatically switch a request from one to the other. The fundamental mechanism is similar for all device types.

Figure 5.
SAA Model Compared with SNA Model



IBM has established firm boundaries between CPI-C and CCS-LU6.2 and between LU6.2 and the Transport Subsystems. Because of these firm boundaries, it is feasible for IBM to specify OSI as an alternate Transport Subsystem. CPI and CCS services, however, would need adjusting in order to hop back and forth between SNA and OSI protocol stacks.

- PrintManager provides the uniform software interface needed specifically by printer resources in order for them to be managed by RRI services and SystemView software.

SAA PrintManager for MVS and VM was announced in September 1990; the OS/400 PrintManager was introduced as part of OS/400 Release 3 in August 1990.

Query Interface (QI)

Query, and the report writing associated with it, is primarily an interactive service that allows users to access and summarize information, and format the results, using menus. In this way, printed results can be obtained without having to write a program.

The SAA Query facility also specifies a program-to-program interface that allows programs to call upon Query facilities. Applications programs can use this interface to manipulate queries, procedures, and report specifications.

In VM as well as MVS/ESA TSO/E, CICS, and IMS environments, the existing Query Management Facility (QMF) base program evolved into SAA Query. In OS/400, no SAA version has been announced yet; eventual support has been affirmed, but no Statement of Direction about the time frame. OS/2 EE includes the query interface in its database management facilities and provides interaction with SQL. The first SAA version ship date was July 1988. OS/2 EE 1.2 Query, shipped in March 1990, added interfaces to C and REXX languages as well as running as an application under the Presentation Manager.

Repository Interface (RI)

The SAA interface to Repository Manager Services for MVS TSO/E was announced on September 19, 1989, when IBM introduced its extension of SAA into the Application Development/Cycle (AD/Cycle) architecture. The Repository Interface is an element crucial to both the proper functioning of AD/Cycle and the extension of SAA into distributed processing. Like a greatly extended data dictionary, it can provide a centralized directory of all the data processing structures available in an organization, centralizing and managing information about data structures so that it is possible to create records and files that can be used by a number of programs. Using the Repository, programmers do not have to create data structures—they can extract them. Repository-related services can result in significant increases in programmer productivity, better management of security, and better cost control.

The information about data structures, file organizations, subroutine locations, and database structures is itself stored in relational databases. Repository Manager Services is a separate set of two software packages (Repository Manager and Dictionary Model Transformer) that create and make use of these databases. The programs are initially available only in MVS TSO/E DB2-based systems, callable from C, Cobol, PL/1, or REXX languages. IBM has made formal statements of its intent to develop Repository Managers on VM (SQL/DS) and AS/400 hosts, with cross-communication and (eventually) file exchanges among all hosts as well as between hosts and workstations.

One of the biggest problems with the Repository as it exists in the context of the new AD/Cycle architecture is that some of the older software embraced by AD/Cycle already has its own data dictionaries and repository-like facilities. For example, packages for Computer Assisted Software Engineering (CASE) from Bachman, Index Technology, and Knowledgeware, which IBM has designated as standard components of AD/Cycle, all have their own data dictionaries at present. From Day One, IBM is faced with the problem of migrating existing software based on potentially incompatible data dictionaries.

Release 1 of the RI was implemented only in TSO/E environments, but Release 2 (September 1990) allowed downloading of a subset RI code to a PS/2 system, so as to provide for a cooperative processing front end.

Resource Recovery Interface (RRI)

The RRI is a new interface supporting resource management by coordinating multiple local resources in such a way as to automatically provide backup when one resource fails. Initially, RRI is implemented only in IMS/ESA TM Version 3 Release 2 environments, which were introduced in September 1990 without a specific delivery date.

AD/Cycle Extensions to CPI

A/D Cycle is an extension of SAA that provides a conceptual framework and compatible tools for managing the ongoing process of applications modeling, design, development, testing, and maintenance of programs. It is oriented toward the fact that before an application can be programmed, it has to be designed. Then, after a program starts running, it may have to be frequently adjusted to changing situations, thus repeating the development cycle to some extent.

AD/Cycle tools involve most (but not all) CPI languages and make use of some CPI services and CUA interfaces.

Whether or not designs are portable depends not only on how they adhere to programming standards but also on how well they fit within the operating characteristics of an organization. Developing portable programs is a two-way street. In order to handle the level of portability involved with distributed processing, some organizations with highly independent divisions might require a degree of Enterprise Modeling that has never really been done for or by them before.

A/D Cycle attempts to provide a framework for this whole process, not just SAA-type building blocks having to do with programming, production, and communications support. In other words, SAA describes tools used in the programming and the execution of programs, while AD/Cycle is involved with control of the background planning, programming, and workflow processes, particularly when they occur in a complicated context. Large enterprises with multiple subsidiary organizations scattered over a wide area may find AD/Cycle particularly helpful, while small, centralized businesses may find it unnecessary.

The extension of SAA into AD/Cycle is a change that involves a new perspective with regard to some of the pieces of SAA. AD/Cycle divides up the world into categories that reposition some SAA products and add quite a few new products and categories that did not exist before. The hitherto unremarkable CSP application generator and the brand-new Repository Manager Interface assume particular significance. Also, most AD/Cycle products are cooperative, requiring a PS/2 workstation "front end" that interfaces to a host S/370, S/390, or AS/400.

Since many products of this type have not previously been available from IBM itself in any form on any product line, IBM has turned to some of its Business Partners to supply the missing pieces. Most are successful third-party products that have been used for MVS TSO/E mainframe environments. Endorsing them within the context of the AD/Cycle architecture means the vendor is committed to

at least some SAA-oriented developments, such as adding graphical PWS-level CUA front ends and interfaces to appropriate SAA languages and services. It appears that IBM will have comparable programs in each host environment, rather than porting the same program through all SAA environments.

Common Communications Support (CCS) Architecture

The Common Communications Support Subsystem forms the underpinnings of distributed processing or cooperative processing with SAA, but it is not the primary focus of most users interested in SAA. This is partly because it is largely seen as emerging out of SNA, so its identity is lost, and partly because it is not meant to be directly accessed by the end user or applications programmer. CPI Services provide access software for the applications programmer, whereas the communications systems programmer is the primary one concerned with CCS.

Regardless, CCS is very important because its limitations curtail the operations of all the levels that rely on it. The strong movement of the market toward function splitting, cooperative and distributed processes, and data would not be possible without adding to the versatility of these communications facilities.

At present, CCS/SNA, together with NetView management functions, represents the most developed environment for complex networks in the world. CCS is SNA's heir, but not exclusively—there are other developments. At this point in time, IBM software for joining remote systems also includes Low Entry Nodes (LENs), Advanced Peer-to-Peer Networking (APPN), Network Nodes (NNs), SNA Distribution System (SNADS), and Open Systems Interconnection/Communications Subsystem (OSI/CS).

SNA, like OSI, uses a seven-layer architectural model. SAA Common Communications Support (CCS) embraces pieces of SNA and APPN as well as providing OSI as an alternative (see Figure 5). How do all these architectures relate to one another?

SNA: Based on IBM's SDLC packet-switching protocol, SNA is a fully developed wide area network with an inherently hierarchical seven-layer structure that enables it to handle large networks efficiently. System Subarea Control Points (SSCPs) in Type 4 Nodes (usually 37XX front ends) operate in conjunction with Type 5 hosts to control the network; SNA Network Interconnect (SNI) software in the Type 4 node provides for interconnection of multiple networks. SNA's LU6.2 program-to-program communications facility predated a similar OSI requirement and is partly responsible for SNA's reputation for "richer" facilities, even though the LU6.2 user base is growing more slowly than IBM would like. Recent changes to SNA have provided for expansions in network size, better peer-to-peer networking within a single subarea, and more flexibility in initiating exchanges. SNA is still expected to govern most of the future development of communications between IBM mainframe product lines, subordinate midrange systems, and programmable workstations (PWSs).

APPN: APPN is an extension of SNA that has been rapidly developing since its introduction in the midrange in 1986, but IBM only just endorsed it as an important extension of SNA in March 1991. APPN uses most of SNA's upper layers and its data link controls, but the transport subsystem routing scheme is different, relying on Control Points (CPs) in Network Nodes (NNs). Its peer-to-peer communications among midrange systems functions in a dynamic, flexible network that is completely independent from the one which connects the midrange system as a subordinate to a mainframe. APPN uses the SNA concept of architectural layers and many specific SNA elements, like SDLC and LU6.2 protocols.

LEN: Both SNA and APPN support peer-to-peer networking of Low Entry Nodes (LENs), a subclass of what SNA calls "Type 2.1" nodes and APPN calls a more limited form of "End Nodes." SNA LENs as well as SNA hosts can interact in peer-to-peer relationships within an SNA subarea or when attached to an APPN network. Within APPN, LENs are capable of the least flexible peer-to-peer interactions.

OSI/CS: The International Organization for Standardization (ISO) has been developing its own vendor-independent, seven-level Open Systems Interconnection network architecture. An architecture is just a blueprint for a real-life network implementation; it was not until IBM introduced OSI/CS (communications subsystem) and OSI F/S (file transfer subsystem) that IBM could say that it had its own OSI network. Previous ISO X.25 data link protocols were used in SNA lower levels, but upper levels are resistent to merged identities.

SNADS: SNADS is a store-and-forward communications facility which is not, strictly speaking, a full-blown network routing facility, but rather a part of SNA, APPN, or LEN. When used in conjunction with APPC LU6.2 program-to-program communications, it resembles a separate network to the point that it is frequently called a "SNADS Network."

SAA's CCS Standards: These standards are both more and less than SNA: they include part of SNA and add OSI networking. Most of the individual facilities are straight out of mainframe SNA (see Figure 2). However, the single most important key to a network's identity, its routing mechanism, is a surprise. It is not the full-blown SNA router, but instead, the LEN node logic which is endorsed in CCS. This brings up a lot of developmental questions about the relationships among IBM network types.

SNA, APPN, and LEN Routing and Network Identity

In view of the differences in capacity between SNA proper and LEN as a separate entity, the current relationship between CCS and SNA is peculiar. LEN, like APPN, uses elements of mainframe SNA in a peer-to-peer networking context, but only between adjacent systems. Since LEN is what has been endorsed, this is equivalent to saying that SAA has not yet endorsed standards which link multiple networks together or which handle routing through one or more nodes operating as intermediaries between two end points.

The addition of APPN End Nodes (ENs) as an SAA standard, yet with full APPN networking (including NN nodes) to OS/2 EE systems, further confuses the issue.

Since SAA's goals involve cooperative processing, distributed processing, and transparencies of all sorts at various levels, communications options must not only be numerous, they must also be flexible, transparent, and far reaching. What happens in a "pure" SAA network when an

Table 6. Callable Interfaces for SAA Service Facilities

Host Environment	MVS/ESA TSO/E	MVS/ESA CICS/ESA	MVS/ESA IMS/VS TM	VM/ESA CMS	OS/400	OS/2 EE	VSE/CICS
CPI-C Communications							
C	•	•	•	•	•	•	•
Cobol	•	•	•	•	•	•	•
DB2	•	•	• *(1)*	—	—	—	—
DDM File Distrib.	—	•	—	—	•	•	•
Dialog/ISPF	•	—	—	•	—	—	—
Fortran	•	—	—	•	—	—	—
PL/I	•	•	•	•	•	—	•
REXX/Procedures	•	—	—	•	•	•	—
RPG	•	—	—	•	•	—	—
Query Management							
C	•	•	—	•	—	•	INA
Cobol	•	•	—	•	—	—	INA
Dialog/ISPF	•	—	—	•	—	•	—
Fortran	•	—	—	•	—	—	INA
PL/I	•	•	—	•	—	—	INA
Repository	•	—	—	—	—	—	—
REXX/Procedures	•	—	—	•	—	INA	—
RPG	—	—	—	—	—	—	—
SQL	•	•	—	•	INA	•	INA
PrintManager							
C	•	—	—	•	•	—	—
Cobol	—	—	—	—	SOD	—	—
Dialog/ISPF	•	—	—	•	—	—	—
RPG	—	—	—	—	SOD	—	—
Repository Manager							
C	•	—	—	—	—	•	—
CSP Generator	SOD	—	—	—	—	—	—
Cobol	•	—	—	—	—	•	—
Dialog/ISPF	•	—	—	—	—	•	—
PL/I	•	—	—	—	—	—	—
REXX/Procedures	•	—	—	—	—	•	—
SQL Database							
C	•	•	•	•	•	•	•
CSP Generator	•	•	•	•	•	EZ-RUN	•
Cobol	•	•	•	•	•	•	•
PL/I	•	•	•	•	•	—	•
Query	•	•	•	•	INA	•	•
REXX/Procedures	•	—	—	•	•	•	—
RPG	•	•	—	•	•	—	—
Dialog Manager							
C	•	—	•	•	—	•	—
CSP Generator	•	—	•	•	—	—	—
Cobol	•	—	•	•	—	INA	—
Fortran	•	—	—	•	—	INA	—
PL/I	•	—	•	•	—	—	—
REXX/Procedures	•	—	—	•	—	•	—
RPG	•	—	•	•	—	—	—

•=Supported.
—=Not supported.
(1) The Transaction Manager is now a freestanding program that can access DB2 and operate independently of the IMS database.

Acronyms:
C—"C" language—not an acronym.
CI—Communications Interface.
CICS—Customer Information Control System.
Cobol—Common Business Oriented Language.
CPI-C—Common Programming Interface for Communications.
CSP—Cross System Product.
CMS—Conversational Monitor System.
CPI—Common Programming Interface.
DDM—Distributed Data Management.
EE—Extended Edition.
ESA—Enterprise Systems Architecture.
Fortran—Formula Translation Language.
I/F—Interface.
IMS/VS—Information Management System.
ISPF—Interactive System Productivity Facility.
MVS—Multiple Virtual Storage.
PL/I—Programming Language I.
REXX—Restructured Extended Executive.
RPG—Report Generator.
SAA—Systems Application Architecture.
SQL—Structured Query Language.
TM—Transaction Manager.
TSO/E—Time Sharing Option/Extended.
VM—Virtual Machine.
VS—Virtual Storage.
VSE—Virtual Storage Extended.

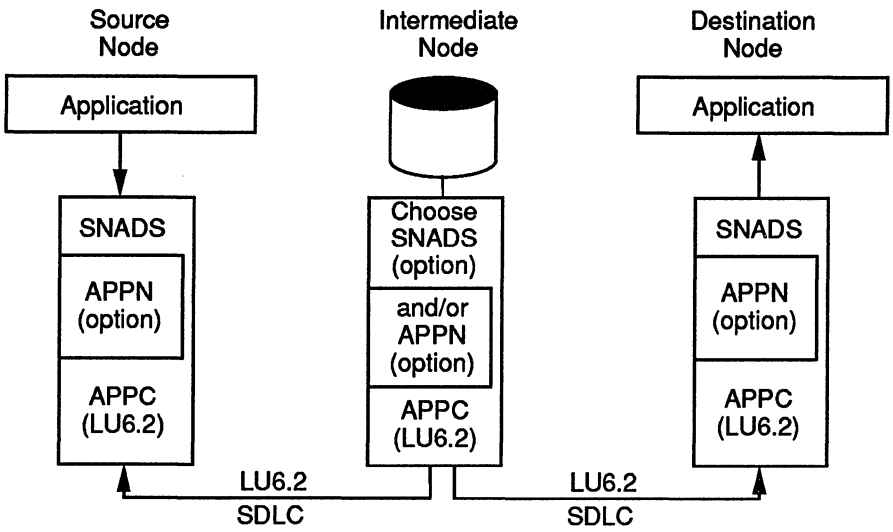


Figure 6.
SNADS Networks

When routing through an intermediate node, APPC and SNADS can be used without APPN or another large-scale network facility even through APPC handles only point-to-point conversations and SNADS does not contain true routing facilities. SNADS logic on the intermediate node stores incoming transmissions on its own disk and then forwards them. If APPN control is added to the SNADS logic, network performance is likely to improve because the disk storage step is not necessary.

application wants to address another application or retrieve a file which is not directly connected but can be reached by passing through an intermediate node?

If the SAA purist must ignore his or her highly versatile SNA or APPN networking software (which, by the way, must be there anyway if the fundamental CCS facilities are there), the only facilities for handling this situation at present are the store-and-forward SNA Distribution System (SNADS), and ironically, OSI/CS. Alternatively, the situation could be redefined by adding a direct line between the end points, making them adjacent.

Adding a line can be expensive, and it also seems ridiculous in cases where a network exists and there is a convenient route through an intermediate node.

It seems clear that IBM wants to endorse a single, coherent network router for both mainframes and midrange, or it would have already set up another double standard by fully endorsing both SNA and APPN. So many parts of mainframe SNA have been borrowed by LEN and APPN that at first it seems that this master network ought to be easy to engineer. However, the thing that is needed to bring the two together here is the same thing that is the key difference between the two networks. It is safe to assume that IBM has something in the pipeline, but whatever the final solution is going to be, it is not one that IBM found earlier.

SNADS

The SNADS store-and-forward communications system is called an "asynchronous" facility by IBM, because it does not require that a communications session be established and then remain open between two end points until communications is accomplished. Instead, a set of "hops" is defined, which can be individually controlled by point-to-point APPC logic. Thus, even though the path passes through several intermediate nodes, each step of the way is treated like a separate communications procedure. The message or file makes one hop and is stored on disk until the intermediate system can arrange the next hop, which may or may not be immediately.

Applications using SNADS on midrange systems that also have APPN can use the APPN logic if they desire. In this instance, a continuously open session would be established, and the passage through the intermediate node would be controlled by APPN instead of SNADS. This may improve network performance, since the disk storage step may be circumvented. (See Figure 6.)

Topographies

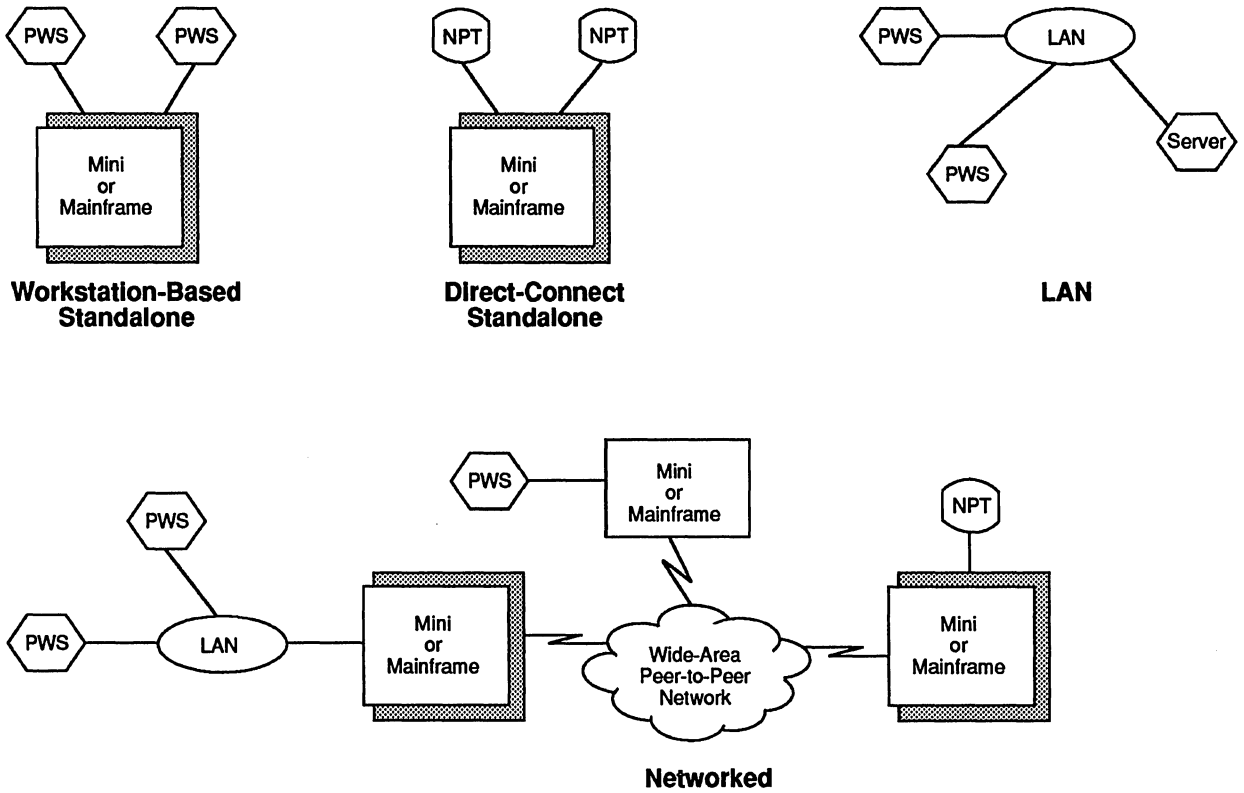
At all levels of the marketplace, multiple processors are being linked together in various ways into a single-system image. Of course, this was always done, in a way, when specialized I/O processors and communications processors were added to a mainframe, but the "single-system image" concept implies that when a user interfaces to what is apparently a single system, multiple main processors and even multiple operating system environments may have been connected together in such a way that they can function automatically as a coherent entity. At all levels, the concept of a single-system image presented to the user is a common element behind concepts of function splitting, cooperative processing, and distributed processing. The use of higher level services of communications subsystems is almost always involved, but the lower level transport subsystems do not necessarily have to be invoked.

The development of versatile intelligent workstations like the OS/2 EE and RISC System/6000 series has been a key step toward the possibility of cooperative processing applications, where programs are split into some procedures which run on the host and others which run on a programmable workstation, but they have not been a driving force until the turning of the decade into the 1990s. The reason is practical: the expensive front-end workstations have to justify themselves. Lowering prices for higher-powered workstations, more and more user interest in the advantages of PC use, and the recognition of the need for better control of those PCs within an organization are the major factors that have brought about market conditions enabling the cooperative concept to become viable.

At the present level of development, the typical division of labor is to use intelligent workstations as an interactive user interface, and host systems or LAN servers for handling shared data or services. The core of the application tends to be on the intelligent workstation. This deduced definition, which seems to be what IBM is producing most commonly at present, implies that any program that makes use of DDM, distributed SQL, or workstations attached by PC Support or Extended Connectivity Facilities or similar products, would be involved in cooperative processing.

Distributed processing is a natural extension of some of the larger cooperative processing applications, but it still needs architectural work before it can take off. Distributed

Figure 7.
Four Key Cooperative Processing Topographies



IBM intends to write its SAA applications so that any one of these topographies can move to any other topography. Distribution of function is presented here; distribution of data may or may not be involved. The Networked and LAN topographies are also suitable for fully distributed applications.

processing among dissimilar system architectures implies enormous standards requirements and architectural developments in order to achieve the capacity to distribute processes as well as data. It has taken IBM almost 15 years to work up the SNA architectural ladder to fully develop the upper level standards, and the OSI organization has not caught up yet. Distributed processing has traditionally been associated with remote data communications systems because this is the context in which it is most likely to be required, although there is no reason that distributed processing cannot (and will not) occur across LAN connections.

The communications/usage patterns resulting from the new cooperative and function-splitting software enabled by SAA are initially falling into several distinct topologies. These are governed by IBM's intention to build software that can be swapped around in both of these respects. With this in mind, OfficeVision products have been designed to fit into four types of usage patterns.

Standalone OS/2 EE-Based LAN Servers: Talking only to locally attached OS/2 EE and DOS workstations on the LAN, these servers are simple two-tiered structures with no remote communications. OS/2 EE systems are "hosts" as well as workstations.

OS/2 EE-Based LAN Servers: These servers talk to other OS/2 EE-based servers as well as their own terminals. This is a communicating two-tiered structure that allows distributed processing and begs for peer-to-peer communications.

OS/2 EE-Based LAN Servers: These servers talk to MVS/XA, MVS/ESA, VM/SP, and/or OS/400 hosts which may or may not be running OfficeVision. Here the LAN server occupies the middle tier of a three-tiered structure, very much like a departmental system. Traditional centrally controlled SNA communications will do quite well, but so would peer to peer.

OS/2 EE, DOS, or Nonprogrammable Terminals (NPTs): These terminals are directly connected to OfficeVision running on a MVS/XA, MVS/ESA, VM/SP, or OS/400 host. This two-tiered structure is like the standalone OS/2 EE LAN structure except that the link between terminal and server can be either a local attachment or a remote communications link. In either case, "direct connection" means a hierarchical interchange between a subordinate terminal and a dominant host.

These usage patterns do not have to be isolated into different networks but may be combined, as shown in Figure 7. They imply interactions between four distinct types of functional elements, each with different roles, which taken together may open the door to new software development patterns in the future. The functional elements are:

Table 7. SAA Distributed Relational Data Access (DRDA) Support

Distributed Environment/ Database	MVS/ESA TSO/E DB2	MVS/ESA CICS/ESA DB2	MVS/ESA IMS/VS TM DB2	VM/ CMS SQL/DS	OS/ 400 DB	OS/2 EE DB	VSE/ CICS SQL/DS
Distributions Can Come from:							
DB2 Host	•	P	P	•	—	—	—
SQL/DS Host	—	—	—	•	—	—	—
OS/400 Host	—	—	—	—	SOD	—	—
OS/2 EE Host	—	—	—	—	—	•	—

•=Supported.
— =Not supported.
P=Partial Support.

Acronyms:
CICS—Customer Information Control System.
CMS—Conversational Monitor System.
DB2—Database2.
EE—Extended Edition.
ESA—Enterprise Systems Architecture.
IMS/VS—Information Management System.
MVS—Multiple Virtual Storage.
SQL/DS—Structured Query Language/Data Systems.
SOD—Statement of Direction for Future Support.
TM—Transaction Manager.
TSO/E—Time Sharing Option/Extended.
VM—Virtual Machine.
VS—Virtual Storage.
VSE—Virtual Storage Extended.

- Traditional hosts, which may be MVS or VM mainframes or OS/400 midrange systems
- LAN servers, which must be OS/2 EE based
- Intelligent PC terminals running OS/2 EE or PC-DOS
- Nonprogrammable terminals

Notice that the OS/2 EE acting as a simple LAN server is a host to its own terminals, but is a Type 2.1 peer to another LAN server. When communicating with mainframe OfficeVision, it can still act as a Type 2.1 node, or it can again shift personality into that of a subordinate terminal.

SystemView as an Extension of SAA C&SM

The foundation for SAA Communications and System Management (C&SM) has always been mainframe NetView. NetView has always been seen as a comprehensive management system, but above all, as a *network* management system. IBM, on the other hand, has realized for years that in order to properly handle system management (configuration management, production and scheduling management, etc.), console management, and network management in a distributed environment, the three had to be integrated.

Furthermore, as distributed processing networks developed in size and complexity, they would be unmanageable unless expert systems and other forms of artificial intelligence were employed to support analyses and sometimes initiate automatic diagnosis and resolution of problems. Automation was viewed as operating in three contexts, as follows. Since all three are equally part of the work load of system management and network management, this model represents a slight alteration of the obvious, or "intuitive," one.

Reactions to Events: This context has to do with message suppression and other tasks related to management of operator activities, regardless of whether the console involved is a NetView console or the system console. This is the "first generation" of automation and in fact has been widely developed already.

Work Management: This context deals with scheduling, distribution, printing, performance monitoring, and storage management.

System Management: This context refers to data capture, prediction of problems, capacity planning, and reporting.

In September 1990, IBM incorporated its network management strategy, its system and network automation strategies, and the related SAA programs into a larger integrated framework called SystemView. SystemView provides a framework for integrated solutions by adhering to yet another patterning, in terms of three management dimensions, as follows.

- The end use dimension: SAA CUA graphics
- The data dimension: SAA file, database, and network data definition standards
- The application dimension: the distinctive SystemView formulation is further divided up into six disciplines, as follows.
  - –Business Management
  - –Change Management
  - –Configuration Management
  - –Operations Management
  - –Performance Management
  - –Problem Management

An important element of the SystemView goal is to open the door for automation. The environment that will be enabled by SystemView is one that allows automated problem determination, automatic responses to problems with

the capability to select alternatives, automatic notification of problems, and automatic restart of the original system when repairs have been effected. Future automation would detect failed devices and automatically route users to alternatives. In the typical manual response of today to break down a component (e.g., a networked printer), multiple users individually discover the problem, call a help desk, and locate an alternative resource until the one normally used is again available.

The intention to integrate system and network management into a compatible, integrated whole that is capable of handling distributed systems underlies IBM's urging its users to automate their system consoles through NetView, not just the network consoles. There are a number of other choices that can be engineered from modules available from IBM, third-party vendors, and the user's own software engineering efforts.

OfficeVision, ImagePlus, and Other SAA Common Applications

In addition to the CUA user interface, the CPI languages and services, and the CCS communications underpinnings, certain SAA Common Applications are supplied by IBM and in some cases will run on all six SAA platforms. These applications, in some instances—like OfficeVision and ImagePlus—are broadly applicable across many types of businesses and industries. Others are more focused on a particular industry, for example, like the SAA applications replacing COPICS, which implement part of IBM's Computer Integrated Manufacturing (CIM) architecture.

The following briefly outlines some important characteristics of OfficeVision and ImagePlus applications. Those SAA Common Applications that are more applicable to particular industries are not included in this report.

OfficeVision

The OfficeVision product family, introduced simultaneously in May 1989 in all SAA environments, was immediately received as the first "major" SAA application. Previously, users were faced with a variety of office systems, including OS/400 Office and its S/3X forebears, PROFS under VM and Personal Services/370 and DISOSS under MVS. The new set of four products—OfficeVision/2 LAN, Officevision/MVS, Officevision/VM, and OfficeVision/400—allowed migration of previously incompatible office systems into a generalized, compatible office system architecture. It also added mouse-oriented pictorial icons to the repertoire of CUA interfaces and provided an example of true cooperative processing using the SAA architecture.

OfficeVision achieves its comprehensive compatibility by exploiting the key SAA architectural attribute: a coherent interlocking code layer is interpolated between the user and the basic individualistic processing. This shell of code turns an identical face to the user, but an adaptive face to the system and the process, adjusting to each specific system's peculiarities, interpreting the icons or other instructions, and routing the processing within itself and/or outward to the local or remote engines which are going to do the actual work. In other words, there is a protocol boundary where a translation occurs between the universal, standardized user interface and the vagaries of individualized processing styles and the location of the actual task execution.

OfficeVision connects programmable workstations (PWSs) and, where possible, nonprogrammable terminals, to local or remote hosts as well as to LANs in order to accomplish local or remote tasks in dedicated or cooperative processing modes. When an NPT interfaces to a midrange or mainframe host, it is clear where all the applications work is going to be done on the host. With a three-tiered system that involves PWSs and a LAN, however, there are an infinite number of ways that a group of tasks can be divided up in order to split the work load between a local programmable workstation, its local LAN server, and a local or remote mainframe or midrange host.

OfficeVision is a modular product, with key modules that are meant to be implemented the same way on each type of system. This allows for a potential "mix and match" approach that can handle a variety of cooperative processing configurations. All four OfficeVision programs, for instance, provide comparable office, word processing, electronic mail, address book, calendar management, library services, and decision support facilities.

IBM anticipates that C language application programs using the OfficeVision application platform will be developed by programmers for a variety of behind-the-scenes functions, such as data conversion between LAN and host, management of variable pools, arrangement of items as elements of a binary tree, and row and column manipulation in tables of variables. There are API "hooks" that programmers need to adapt the system to different languages and environments, which, for instance, might expect sequences, sorts, searches, and insert text procedures to operate in different ways or might require that date and time values be managed differently.

Capabilities specific to OfficeVision as an office system will continue to be added in all environments in an upward-compatible manner; there is no need to elaborate further on more of them.

However, there is another aspect of OfficeVision that is of interest to all current and future SAA applications; that is, its capability to act as a model and a vehicle for other distributed solutions. As a model, it shows certain relationships between terminal and hosts, and certain topographies involving LAN-based workstations and several other hosts.

OfficeVision provides services to a directly attached nonprogrammable terminal, a directly attached OS/2 or DOS-based programmable workstation, or a local or remote OS/2 Extended Edition LAN server. From the point of view of the terminal user, there are three types of connectivity orientation governed by these terminal types.

PWS Requestors: OS/2 EE or DOS workstations can operate as requestors when attached by means of a LAN to OS/2 EE servers or when directly connected to MVS, VM, or AS/400 hosts.

NPT Requestors: Nonprogrammable terminals (NPTs) can directly connect as requesters to host systems, but not to LAN servers.

All-Purpose PWSs: OS/2 EE workstations can do everything, depending on whether they are defined as requestors, servers, or nonprogrammable terminals. Host connectivity is more complex. At its current level of development, OfficeVision software components gravitate toward four different types of topographies, based on communications usage. Three out of four depend on how the OfficeVision/2 LAN host (requester/server) relates to other hosts. Large networks can mix these types of connections, of course.

Table 8. Basic SAA and AD/Cycle Components Compared

SAA Categories and Components	AD/Cycle Categories and Components
Common User Access (CUA) Models	
• Programmable (PWS) entry	—
• Nonprogrammable (NPT)	—
	Application Development Platform
• Graphical PWS CUA	• Graphical PWS CUA
• Workplace	• PWS: PDG/WSP
	• Tool Services: PDF-SCLM
	• AD Information Model
SAA CPI Services	
• Repository Interface	• Repository Services: —Repository Manager —Dictionary Model Transformer
• SQL DRDA, Query DB Interfaces	—
• Dialog, Presentation Interfaces	—
• CPI-C Communications Interface	—
CPI Languages and Generators	**CPI Languages and Generators**
• C, Cobol, Fortran, PL/1, RPG	• C, Cobol, Fortran, PL/1, RPG
• REXX Procedures	—
	• Inspect (for C370 and PL/1)
• CSP/AE Runtime only	• CSP/AD Development, Runtime, PWS
	• CSP/AD External Source Format (ESF)
	• CSP/370 Runtime Services (RS)
—	**Knowledge-Based Systems**
	• Knowledge Tool
	• Expert System Environment (ESE)
	• Knowledge Engineering Environment (KEE)
—	**Test/Maintain Tools**
	• IBM Workstation Interactive Test Tool
	• IBM Software Analysis Test Tool (SATT)
	• Cobol/SF and CCCA
—	**Enterprise Modeling Tools**
	• IBM DevelopMate
	• KnowledgeWare Information Engineering Workbench/Planning (IEW/PWS)
	• Index Technology Corp. PC PRISM
—	**Analysis/Design Tools**
	• Index Technology Corp. Excelerator
	• Bachmann Information Systems Re-Engineering Product Set
	• KnowledgeWare IEW/Design (IEW/DWS)
	• KnowledgeWare IEW/Analysis (IEW/AWS)

Standalone: OS/2 EE-based OfficeVision LAN Servers can be standalone units which may or may not attach local OS/2 EE and DOS workstations on a LAN. Neither the standalone nor the LAN configuration has to involve outside communications.

LAN to Host: This refers to OS/2 EE-based OfficeVision LAN Servers talking to MVS/XA, MVS/ESA, VM/SP, VM/XA, and/or OS/400 hosts which are running either OfficeVision or the precursor appropriate to that system. Here the LAN server can be a standalone system rather than a server, or it can sit between the host and other workstations on a LAN, occupying the middle tier of a three-tiered structure. Since the mainframes can support them, any and every type of communications topography is to be expected.

LAN to LAN: These OS/2 EE-based OfficeVision LAN Servers talk to remote OS/2 EE-based OfficeVision Servers as well as to their own LAN-attached terminals, if any. Since OfficeVision is to be the same on every system, the remote LAN server is functionally indistinguishable from a mainframe host. However, this communicating structure invites distributed processing and LU6.2 APPC interchanges because the limitations on the power and complexity of the OS/2 systems invite them to spread the work around.

Direct Connect: Direct Connect refers to OS/2 EE, DOS, or nonprogrammable terminals directly connected to OfficeVision running on a MVS, VM, or OS/400 host, without the real or implied existence of a LAN. The link between terminal and host can be either a local attachment or a remote communications link. All OfficeVision processing is done on the host, by definition, and the terminal is treated as an NPT regardless of what it really is. If a locally attached OS/2 EE does some of its own processing, it ceases to be a direct connection and becomes the first configuration described above.

The optional character of OfficeVision cooperative processing, combined with the mirrored capabilities ensured by SAA, opens some interesting doors. IBM talks

about the host doing processing for the workstation or for the LANs, but not much about the possibilities in the opposite direction.

The increase in database capacity, Token-Ring performance, and PS/2 power, coupled with the mix and match potentials of the OfficeVision design, mean significant expansion in LAN power and flexibility. It should be possible to off-load the OfficeVision mainframe onto the LAN as well as vice versa. The OfficeVision LAN is not unlike a timesharing multiprocessing mainframe. Although a standalone LAN system is really a networked group of PCs, it is treated as a single system, since other computers see only the key server address.

Will such LANs prove to be alternatives for midrange systems? Will the LAN work well as a server for the mainframe? Instead of expanding mainframe capacity, how about expanding LAN capacity? Many PC users are convinced that ultimately this will be the case. Certainly, the addition of full-scale APPN networking and improved network management make it more and more possible to treat a LAN as if it were a type of networked multiprocessor, a single entity on a peer with midrange and mainframe hosts.

ImagePlus

Next to OfficeVision, the hottest-architected SAA end-user application products are probably the ImagePlus set of programs, designed to support not only the processing of images but also the combination of those images with data and graphics.

Unlike office automation, image processing is a leading edge technology, closely related to other leading edge technologies such as optical scanning and optical storage, which are only beginning to find their place. The increased power and memory capacity of workstations with front-end capabilities, like the PS/2 and RISC/6000; industry interest in desktop publishing; and the development of the cooperative processing modification of the client/server program model are indications of how ripe the times are for image processing. However, prices must drop to make the technology more affordable and widespread, particularly with regard to laser printers, optical scanners, and optical storage.

ImagePlus handles documents not only as individual documents but also as part of "folders" and "cases." Processing of documents and folders can involve capture using scanners, indexing, storage, and retrieval, but folders involve larger groups of documents. Case processing involves a more automated approach, with groups of documents managed in queues that allow time-managed or stage-managed workflow.

Like OfficeVision, ImagePlus applications are cooperative, using an SAA/CUA programmable workstation for much of the front-end image processing and an SAA host for a back-end server and processor. At present, hosts can be MVS CICS/ESA, MVS IMS/ESA TM, OS/400, or PS/2 (LAN) systems; workstations can be DOS based as well as OS/2 EE based. There are no VM/CMS or MVS/TSO implementations.

In September 1990, the introduction of the second wave of ImagePlus resulted in the following product sets.

- SAA ImagePlus Folder Application Facility: MVS/ESA Version 2
- SAA ImagePlus Object Distribution Manager: MVS/ESA Version 2
- SAA ImagePlus Object Distribution Manager/400: OS/400 Version 2; includes folder handling
- ImagePlus Workstation Program/2: Version 1.1.0 (OS/2EE 1.2.0)
- ImagePlus Workstation Program/DOS: Version 1.2.1 or 1.2.2 (with DOS 1.4.0)

At the mainframe level, MVS Version 1 ImagePlus required an IMS database, but in Version 2 images are stored in DB2 local or distributed databases. The user is expected to choose between a CICS or an IMS TM transaction manager (TM no longer requires IMS itself). There has not been an ImagePlus implementation for either VM/CMS or MVS/TSO/E yet.

When OS/400 systems are used as servers, the ImagePlus Workfolder Application Facility/400 Version 2 requires the PC Support program. Version 1 supports only DOS-based workstations, while Version 2 supports both DOS- and OS/2 EE-based workstations.

ImagePlus systems can also be wholly LAN based, using a PS/2 server, but the large amounts of storage needed to support image handling and the large documents, folders, and even "books" that are likely to be involved with ImagePlus capability suggest that OS/400s or mainframes are the more likely servers. Formal "Statements of Direction" (SODs), released in September 1990, emphasized IBM's commitment to making APIs on all SAA systems consistent with one another and to supply a variety of optical storage products.

SAA, OSI, and AIX

For the most part, this report has focused on SAA as it is implemented within IBM's own SNA mainframe environments and APPN midrange peer-to-peer extensions. SAA, in the interests of multivendor networking and interoperability with other architectures, has been endorsing other networks and bridges as well. The most important of these is the OSI communications code stack, which will provide SAA CCS alternatives to SNA and APPN.

Initially, the OSI/CS subsystems are just as monolithic as SNA. Although IBM wants to create points of exchange between SNA and OSI code stacks in the future, at present, when a program is slated to use OSI, it can use only OSI services and transport. Furthermore, the transport level is very undeveloped, since it consists of the X.25 NCP Packet Switching Interface (NPSI), which basically does nothing more than hitch an IBM system to a non-IBM X.25 network. Although there is a new store-and-forward program quite like SNADS, IBM at present does not provide software that creates a genuine OSI networking backbone, although some programs (like XI) look that way at first.

As a point of clarification, AIX is not at all architecturally comparable to OSI. OSI is a networking architecture, like SNA, APPN, and TCP/IP. AIX is a UNIX-type operating environment, comparable to MVS, VM, OS/400, or OS/2. What IBM clearly intends for AIX is not full SAA compatibility, but a distributed processing role that will make use of SNA, APPN, OSI, or TCP/IP communications as user needs dictate. IBM has asserted in its position papers for the RISC System/6000 that if there is a conflict between SAA compatibility and AIX compatibility, the decision will be for smooth upward AIX growth through AIX UNIX compatibility. Nevertheless, there have been several SAA compilers and supporting system software

and compatible communications links supplied for AIX. Since the AIX environments must retain compatibility with UNIX standards to remain compatible in their market niche, SAA compatibility will rely more on bridges and converters than VSE systems will. "Viaducts" have already been released for AS/400 and MVS systems.

Compatibility

SAA allows users to consider some new scenarios. Previously, and even now, the relatively low-cost 9370 and low-end ES/9000 systems invited users to unburden mainframes of visible backlogs and to computerize the hidden ones, because they are completely compatible systems with a lower cost per MIPS. The real savings are in the incompatible architectures—the PS/2s and AS/400s. The invitation to download is made especially tempting by the increasing availability of third-party application enablers that help the user alter or trap host code intended for non-programmable terminals and send it to PS/2 cooperative workstations that can do much of the work previously done on the host. These application enablers have been in existence for several years, so it is now quite clear that adding cooperative processing by front ending an existing mainframe program can indeed be done with little or no changes to the existing host program and yet be very cost effective.

Some organizations try to prevent compatibility and data integrity problems by authorizing acquisitions of only certain hardware and/or software products, but this controls only some problems. Even rudimentary distributed processing, hardly worthy of the name in many cases, can create some knotty data integrity problems that must be controlled if distributed processing is going to work efficiently. Data centers cannot really prevent the acquisition of inexpensive PCs and thus must find ways to deal with many small but growing pools of data over which they have had little or no control.

Many distributed processing applications are described in a top-down fashion: such and such an application is split into two, with the implication that all of it need not run on the larger system. Gaining control of these databases and processing pools is like implementing distributed processing from the bottom up, where the mainframe starts to participate in and control an activity that previously was solely the domain of the workstation. The proliferation of PCs in business, and the resulting uncontrolled data processing/data storage pools, is the primary situation driving not only the move toward control of distributed databases but also the move toward control of distributed processing.

Compatibility Problems with Individual Programming Standards

Because the Common Communications standards elements involve either the lower levels of SNA, the ISO OSI communications architecture, or application-independent logical entities, all of which are usually characterized by distinct protocol boundaries, maintaining agreed-upon compatibility standards is less of an issue than it is for the Common Programming Interfaces. Although many users have non-SNA applications that they must maintain for various practical/historical reasons, and despite IBM's somewhat different implementation of SNA on different systems, the basic concept of the logical separation of SNA software as a different sphere (with its own set of separate experts and managers) is a concept which supports the acceptance of Common Communications as a workable and separable standard. When the large network routing problems for mainframe SNA, APPN, and OSI are resolved, they will be incorporated in the system code, and most users will find all application code will be unaffected.

The Common Programming Interface is different. As we described earlier at greater length, so-called standard, high-level language compilers differ from system to system as to instructions available and results produced. User programs may necessarily involve system-dependent characteristics (as in the case with the approximations involved in many floating-point operations), or low-level code can be deliberately included to improve performance. Even the system code of the PC product lines, ASCII, differs from the EBCDIC code of the IBM mainframes and midrange systems. If portability is to be workable, it is obviously important that a programmer/analyst isolate those operations that are "basic" to all SAA levels and those that are "extensions" for use only in some of the environments.

The SAA standard calls for openness. The practical result is that IBM publishes a special set of SAA manuals particularly oriented toward compatibility. An Overview manual, first released in May 1987 and updated once or twice yearly, provides comparison tables for every language that are helpful in making general distinctions for the basic Common Programming Interfaces. However, it is the individual manuals on each standard that spell out the details needed by programmers. These SAA-oriented manuals complement rather than replace the basic manuals. They contain detailed descriptions of specific operations, data organizations, etc., particularly in regard to their portability and their relation to ANSI and other standards. An easily distinguishable green print highlights information on instructions that are implemented in one environment and not another, that have a different syntax (code format) in different environments, or that involve different semantics (behavior at run-time). The manual also sets apart information on each of IBM's own extensions to ANSI standards by boxing it in a second section immediately following the explanation for the related standard operation. Programmers used to the basic function can easily see how and when to use the extended facilities.

Levels of Compliance

What does SAA mean for operating environments that already include some, but not all, of the existing programs that make up SAA—for example, VSE/ESA, MVS/370, and PC-DOS—and yet are not designated as full voting members of SAA? All of these environments, and more than a few others, are full participants in SNA/SAA Common Communications and include many Common Programming Interface elements.

IBM explains that designating an SAA system as fully compliant implies a commitment on the company's part to deliver a full set of SAA elements to each SAA environment, but this does not mean that an undesignated system cannot participate. Of course, a backdoor SAA user would have to do a bit of sleuthing in order to gather the information that is currently compiled in manuals for designated SAA systems, isolating differences in software functioning in different systems. The easy-to-compare documentation would not be available for the non-SAA system.

On the other hand, no user operating in an environment designated as a full voting member of SAA has to implement every element of the standard in order to create an SAA program. Most applications need only a small part of the SAA product suite. They must use an SAA compiler and adhere to the Common User Access guidelines. If the

program is to communicate, only SAA elements are to be used. Whether it is practical for the user to code in missing elements depends on the program type and the communications or database access involved. Since ANSI standard compilers like those used in SAA are plentiful, the really distinguishing core of minimal SAA is not a program at all, but the handbook that describes the required end-user interface.

It seems clear that the nature of the SAA architecture and its detailed documentation is such that outside vendors could be partial participants in much the same way as IBM's own non-SAA product lines are. In time, as applications software becomes more widespread and the standards are better known, a competing vendor might even comply fully with SAA. After all, this is an architecture designed for portability across incompatible systems, and IBM is saying it is committed to an Open Architecture.

IBM versus Digital

Some observers say that SAA is an attempt to match the easy portability of Digital's VAX software, but that analysis overlooks the individualistic nature of IBM product lines. It is more realistic to compare Digital's VAX environment exclusively with IBM's VM environment and think of the rest of IBM products as if they come from different vendors that happen to share the same communications architecture and marketing and support facilities. (Of course, IBM does not look at it that way, although sometimes it hints at the idea.) In the VM environment, IBM software products, just like Digital programs, are now easily portable upward with little or no recompiling or converting, although some benefits are lost at the XA/ESA levels without a recompilation.

Pandora's Box

Day-to-day data and hour-to-hour conversions, interactions, and interfacing integrity are not the only compatibility issues for SAA. Let us suppose a distributed processing system operating over a peer-to-peer network consisting of a few (three) 3090 mainframe complexes, ten 9370s, thirty AS/400s, and one thousand PS/2 systems. Will all 1,043 systems have to be updated at once, every time a significant change occurs to the mainframe SAA capabilities? Otherwise, will the system/network management facilities keep track of each system configuration, the implementation levels on each system, and which application programs expect what level of implementation, in case a program is ported elsewhere? What about the impact of all this management traffic on overall network performance?

IBM acknowledges that ideally, as SAA develops, alterations and upgrades to SAA elements are made in all environments concurrently, and new developments in the area of change management are definitely needed. As much as IBM would like to institute change, the development backlog means that global, concurrent changes remain impossible for some time to come. Complex issues like these make users more willing to abandon a feisty, do-it-yourself attitude in the hope that Big Blue will come up with another grand plan that provides an organizational backbone.

Clearly, SAA itself involves product development cycles which could benefit from AD/Cycle control and SystemView management. ■