Integrating applications with SAA

by L. A. Buchwald
R. W. Davison
W. P. Stevens

Advances in computing technology and reductions in development cost have greatly increased the number of people who use computers, and have expanded the number and types of applications available to them. People want their applications to share data and to be consistent with one another with respect to terminology and appearance. They also frequently need access to applications and data on computers in other locations; the computers may be models and types that these persons do not normally use. Integrating application functions in a seamless environment is an important step toward satisfying some of these requirements. This paper discusses what integrated applications are, why they are valuable, and how Systems Application Architecture (SAA) can make it easier to develop them.

Integrated applications are diverse functions and services that work together and look like one consistent system to the end user. They coexist in an environment that provides a framework for promoting consistency of appearance and terminology. Such an environment would allow applications to share services and exchange data in a standardized fashion. It would also allow users to switch between applications as required by the job being performed.

Computer networks make it possible for users to store needed applications and data in many different places. Figure 1 illustrates a simple configuration with an end user and two different applications. Each of the applications is located at a different node in the network and executes in a different operating environment. If the applications were integrated from an end-user point of view, they would appear to be running in a single environment close to the user, as illustrated in Figure 2. The user would be able to switch back and forth between the applications, with the system transferring data between the applications as required.

Value of integrated applications

During the past few years there has been a significant growth of new business applications over the broad range of IBM hardware, a trend resulting from the need of business professionals for increasingly accurate and timely information to help run businesses. This trend is also the result of an improved ability to deliver computer applications.

The information center, for example, has assisted business professionals with applications and decision support tools that help them do their jobs more quickly and accurately. Another example is the computing power delivered by the personal computer. Business solutions may now be implemented and tailored applications developed independently of a company's computer experts. The personal computer is ideal for business solutions that are highly interactive and must react to the individual user's needs with tailored output. Because of their former high cost, such applications as graphics design, spreadsheets, and complex document formatting were previously available to only a few users. Many of today's business applications, however, run as self-

Figure 1 Applications around a network

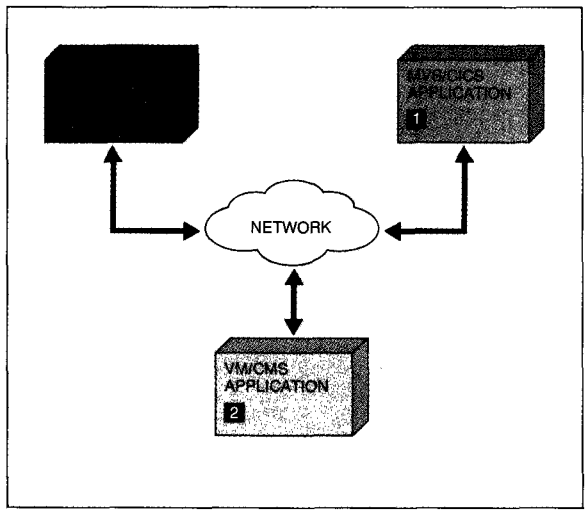
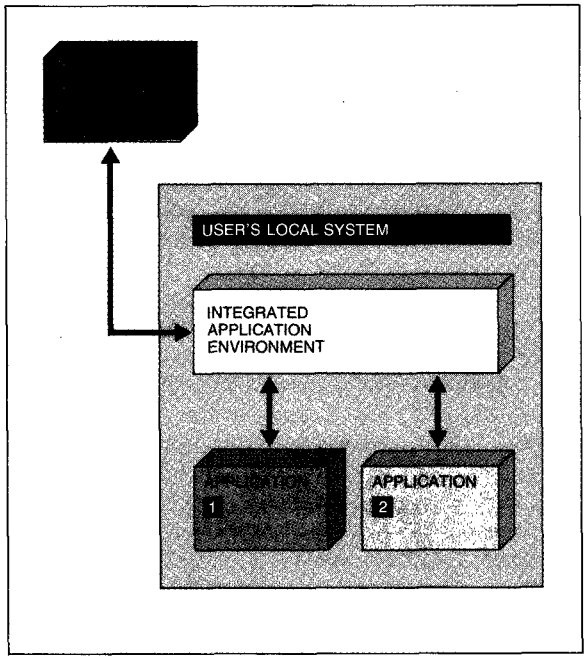


Figure 2 User's view of integrated applications



contained, independent units that do not interact with one another or with the user in a consistent manner.

The growth of new business applications has increasingly obligated the user to remember multiple ways of accessing and interacting with different operating systems. Specifically, it requires familiarity with different environment and application command sets and interaction techniques. A mental adjustment may often be necessary to accommodate the new environment or application each time the user switches applications.

In order to take advantage of the productivity gains provided by new business applications, the applications must be available to work together in a consistent fashion across different hardware and operating environments. In particular, a technique must be provided for switching applications (i.e., moving from one application to another and back in a spontaneous fashion), with no need to explicitly terminate one application before starting another.

To present more fully the user's view of integrated applications, Figure 3 illustrates some of the shortcomings of nonintegrated applications. The figure might exemplify a user in the insurance industry who has responsibility for claims processing. The claims-processing transactions are implemented on an MVS/CICS system, and PROFS™ office applications are available on a VM/CMS system. The user follows a standard sequence of steps for processing each claim, but may have to deviate from the normal sequence of events to respond to an abnormal situation. Consider the case in which required information necessary to complete the claims process has not been received from the claims adjuster. When this is discovered, the user must send a memorandum or note to the adjuster inquiring about the status of the investigation and requesting the missing information. To accomplish this, the user must switch from the claims-processing application to the PROFS office application. Coincidentally, the user expects the claimant's name and claim number and the adjuster's name and electronic mail address to be transferred automatically to the PROFS note application. The user wishes to view the two applications as though they were running together in the same environment.

In today's typically nonintegrated applications, the user most often sees applications as shown in Figure 3. It is necessary to switch between two different computing environments each time there is a need to use the other application. The claims processor may have to take the following steps:

- Write the necessary information to be transferred to the note application.
- Wait to terminate the claims-processing application gracefully.
- Log off the CICS system.
- Transfer via VTAM or a hardware switch to the VM system and log on.
- Start PROFS and the note application.
- Manually enter the data to be transferred, as well as any text, to the note application.
- Send the note.
- Return to the claims-processing application through a similar procedure.

Integrated applications, on the other hand, can provide a consistent user view by masking the complexities of system access, the dependencies on application location, and the differences in data formats. From the user's point of view, integrated applications are consistent in screen layout, terminology, commands, and navigation, and they can easily access and process data from other applications. Integrated applications are easy to use, both individually and collectively, and they do not have to be terminated in order to switch from one to another.

The chief user benefit of integrated applications is improved productivity, which is manifested in several ways:

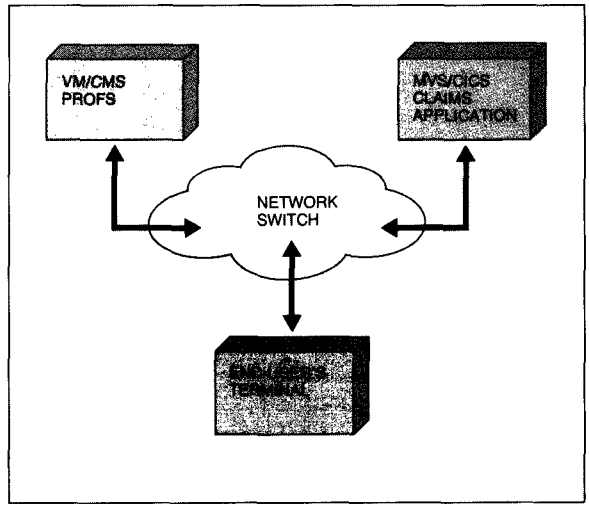
- Users can transfer computer skills from job to job with minimal training. They can learn the new job-related applications more quickly and easily because of common methods for application execution and interaction. Education and training are limited to learning new application functions and their relationship to job-related tasks.
- Application execution and interactions can be tailored to the user's skills and job requirements. Application functions may be sequenced to support the job at hand, rather than tailoring the job to satisfy computer system requirements or restrictions.
- Application location is transparent; the user only requires access to a computer system to use the applications.

It is the intention of SAA to provide a framework in which applications can be developed and integrated in these ways.

The integration of applications

Application integration today most often occurs between functions within a single application or between functions of related applications. Applications may be related by industry or by product family. Current and future application development should aim toward the integration of unrelated applications, such as those shown in the claims-processing and office examples provided in the previous section.

Figure 3 Using applications in different environments



Designers must address both planned and unplanned integration. Planned integration is achieved when all the applications involved are intentionally integrated by design and implementation. Unplanned integration comes about when functions are used collectively among applications in a manner that was not intentionally planned for at the time the applications were developed. This includes the need to integrate new applications with applications that already exist. It also includes the spontaneous requirement to specify application interactions that are not known at development time.

SAA provides a number of facilities to help developers integrate applications, including the Common Programming Interface (CPI), such as those described in SAA for database, data access, and languages. Other facilities are provided through underlying product functions and common architectures supported in all SAA environments. It is important to remember, though, that SAA architectures and products are continuing to evolve. Additional elements and functions may be provided as requirements for them are identified, as solutions are defined, and as resources are committed for all SAA environments.

Sharing data

Seemingly unrelated applications frequently require access to the same data; a good example of this is an employee address file. Data stored in such a file may be used by payroll, personnel, the employee stock plan, expense accounting, and other applications.

Today, users frequently require access to applications that run on different systems to take advantage of unique hardware features.

These applications, however, are typically developed separately, usually in different groups organized along functional lines. The result is different versions of the same data with slightly different content and structure, each of which is created, maintained, and accessed separately.

There are several ways applications can share data. They can access shared files or databases, pass data to other applications, or share global data in memory. However, the sharing of global data in memory is not recommended, because it results in inflexible, complex applications and prohibits distribution of the functions to other computers in the network.

Sharing data can be difficult because data formats are often unique to each application. The SAA Data Content Architectures, data access techniques, and system facilities address this problem. With these architectures, applications may be "educated" to understand the data that are received and the processing to be performed. In this manner, data may be shared in a single computing environment or across multiple computing environments.

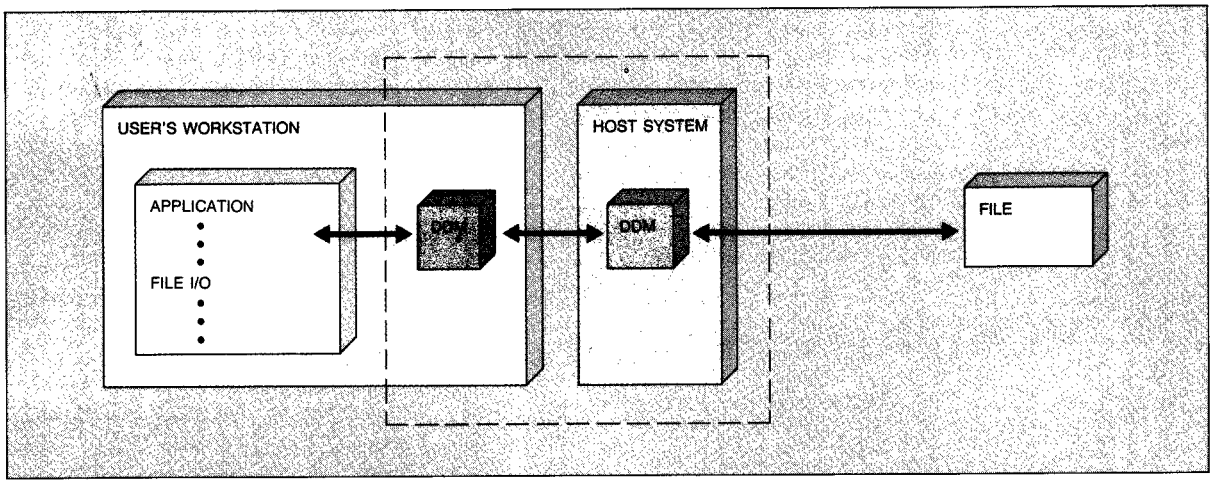
Shared data. File access methods and database managers allow concurrent accessing and updating of shared data. Multiple applications using the files need not be in the same execution environment. In addition, database managers provide a level of transparency between the applications and the data. A database manager allows for storage layouts to be changed without requiring changes to existing applications.

SAA further simplifies data sharing across environments. An SQL interface is provided in all SAA environments as a common access technique and programming interface. This allows applications that are developed independently to share data in a planned way. Databases can be designed to accommodate a range of application requirements, with individual applications defining unique views to accommodate specific requirements. New applications can add data and share common data with applications developed earlier. This also simplifies support of on-demand access to shared or common data with generalized applications such as queries and spreadsheets.

Remote data access. Today, users frequently require access to applications that run on different systems in order to take advantage of unique hardware features. However, it is often not practical or convenient to maintain multiple copies of data at different sites to facilitate the sharing of data among the applications. Data are frequently stored where they may be most easily maintained and made accessible to the largest number of users, though this approach is not necessarily convenient for all users. An example of this is the requirement to process data stored in a host connected to Intelligent Workstation (IWS) hardware. Consider a user who has an interactive business graphics application, an all-points-addressable display, and a plotter that is available on the workstation. Traditionally, the user must perform a series of steps in order to abstract the required data from the host database and download them to the workstation for tailoring or reporting. However, if the format and content of the host file can be described to the business graphics program, the file can be accessed as though it were a local file. The transparent access to host-stored data is supported by the Distributed Data Management (DDM) architecture and products defined as part of SAA.

The workstation program opens a file as though it were a local file. When an OPEN command fails, DDM intercepts the failure and attempts to find the file on the attached host system. If the required file has been defined to the workstation with the proper access and authorization, all subsequent READ and WRITE commands to the file are routed to the DDM component on the host, and the records are passed between the file and program as though both resided

Figure 4 Accessing files via DDM



in the same computer. Figure 4 illustrates this design as it appears to the application program and the user. The components contained in the dashed-line box appear transparent. For simplicity, some of the components such as the communications components are not shown.

Data interchange. Today, applications in different address spaces or execution environments can communicate, but the techniques are complex. Higher-level communications access methods can simplify this process by allowing applications to send and receive messages and data by identifying the recipient and using common communications techniques and protocols. SAA provides architectural definition for products for data interchange. This includes exchanging messages using common communications support, passing parameters using a higher-level call interface, and controlling data passage via application management functions (as, for example, in the OS/2™ presentation manager) or dialog manager services.

In addition, high-level functions are being developed that allow each end of the interchange to request and recognize changes in the communication flow. Standardized interchange data structures and formatted data streams are being architected to reduce application code dependence on fixed data formats. The self-describing characteristic of architected data streams facilitates data sharing among functionally diverse applications that may be implemented at different times. Thus, applications can recognize and react to changing conditions and requests.

Direct interchange of data messages is supported by SAA through a programming interface, a common set of delivery functions, and a set of data-stream architectures. Programs that communicate using these facilities can be developed independently of one another. This allows for generalized application functions such as memo and mail to be used on demand by other applications, with no special programming requirements. The requesting application would need to know the name and data requirements of the function being requested.

SAA also supports data interchange between applications via variable pools and services provided in dialog services. Applications designed to use the variable pools and variable names can access and update information stored there by other applications. Similarly, data interchange is supported by the presentation management component of SAA. The interchange is under user control between the screens of currently active applications, where, to the receiving application, it appears as though the user had entered the data via the keyboard.

In summary, using dialog services and presentation services, application front-ends on the workstation can facilitate data interchange between independently developed applications available on different hosts. The next section presents additional discus-

sion of this approach, under the heading *Executing remote applications.*

Sharing functions

SAA dialog services provide a common set of facilities to support and manage the dialog between a user and an application and between independently developed applications. SAA dialog services allow applications to invoke each other in a standard way. A common set of facilities provide a structured approach for menus and displays to be developed and maintained independently of application code; they also provide an invocation mechanism which is consistent across environments and applications.

It is valuable to design applications to be portable from one processor to another.

Applications developed using SAA dialog services as an invocation and display interface will, independently of the application code, simplify the tailoring of menus, panels, and terminology to specific installation or individual requirements.

The sharing of application function in SAA is not limited to using dialog manager or application functions executing in the same computing environment. SAA defines a common set of SNA Advanced Program-to-Program Communication (APPC) and a Common Programming Interface (CPI). The communications CPI allows programs to talk to one another, and it provides facilities for applications to be developed that can start their own function or start shared function on another system.

Portable applications. When an enterprise relies on multiple processors (that may be of different hardware architectures) to meet its information processing needs, it is valuable to design applications to be portable from one processor to another. The benefits that may result are no need to rewrite applications, consistency from one system to another, and more timely delivery.[1] Another reason is to facilitate application integration. Remote sharing of data, data interchange architectures, advanced communications, and other advances in technology will help simplify application integration across environments. Integrating applications may be most simplified when the applications are stored and executed in the same environment, or the primary environment of the user.

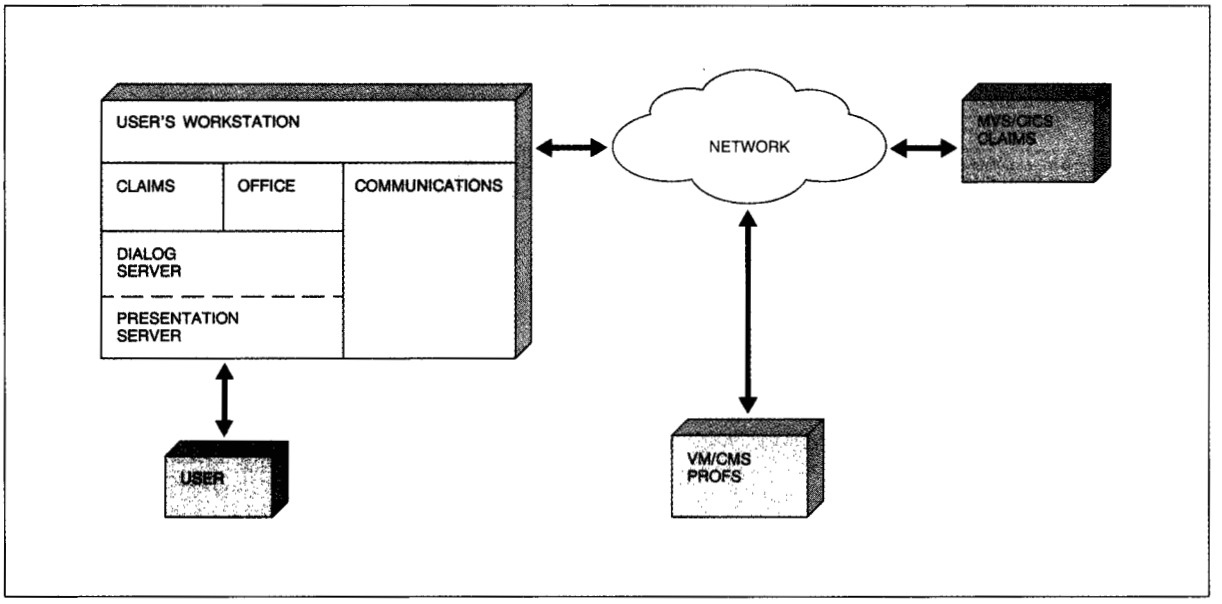
Application design and development for portability must be carefully considered. Modularization is a very important technique in achieving application portability. Specifically, the use of structured concepts to separate functions is what allows the reuse of common functions and facilitates portability. Integrated applications should perform the same function consistently, even after maintenance to those functions. The productive way to accomplish this is to have the sharable functions as modules included in each using application. When separating the functions, it is important to strictly separate I/O operations from modules containing the business processing logic. This is a valuable procedure because the logic is the most easily ported component, and the I/O is most likely to require change and probably the easiest to rewrite.

SAA languages permit the exploitation of facilities unique to specific environments, applications, or parts of applications. Applications which are intended to be portable should use the SAA language subsets and facilities which are common across environments. Modules which exploit environment-specific facilities should be isolated for ease of portability.

Common Programming Interfaces (CPIs) will be defined for all SAA elements. Thus, it will be easier to build applications for distribution throughout the network on the basis of integration and resource usage requirements. It will also be possible to redistribute the applications as requirements change. This can be contrasted with implementation decisions that were dictated by the availability of functions in a single environment.

Common application functions. SAA common application functions are usually general functions that apply to more than one business. They may also be functions specific to a particular business which are used by other businesses. Many office functions, for example, are considered to be generalized functions. Applications using general functions are more port-

Figure 5 Using application front-ends in workstations



able, because these functions will be implemented with the same interfaces in each SAA environment. The SAA common application functions will also present a consistent appearance to the user and may be executed remotely.

Using office mail to distribute reports in a manufacturing application is an example of integrating applications by using general functions. Another example could be that of using the office memorandum function to add address information to correspondence generated by an insurance claims-processing application.

Executing remote applications. SAA provides the interfaces necessary to take advantage of the one SAA environment (i.e., OS/2) that has application management functions. The presentation manager in OS/2 supports application management functions that are extensions to the SAA presentation interface. Applications written in any of the SAA environments can take advantage of these application management functions by using the SAA communications interface to connect to an OS/2 workstation. An application written in this cooperative fashion has its user interface running on the OS/2 workstation. The remainder of the processing takes place on the host system.

A common front-end (or multiple unique front-ends) can be written for the workstation using OS/2 presentation services and dialog manager services if desired. By utilizing the SAA communications component and communications CPI, the front-end can invoke the main application function on the host. In this manner, SAA support allows the user to connect to multiple applications running in different hosts concurrently. Figure 5 illustrates this approach to application integration, using the earlier example of an insurance claims processor accessing two diverse applications concurrently.

With this approach, the user has a consistent view of application invocation. A common front-end utilizing application management function in OS/2 presentation services could present the user with choices of applications. The user would not be aware which applications run locally and which run on the remote hosts to which the user is connected. Application switching would occur when the user transferred between the application windows.

When a cooperative application is started, the front-end establishes a communication session with the rest of the application in the appropriate host. The user need not be aware of the steps that are involved

to perform authorization and initialization of the host portion of the application. The work to accomplish this is contained within the local and host components of the application, using the security facilities available in each system.

Common User Access

The SAA Common User Access (CUA) publication[2] defines standards and guidelines for the appearance of, interaction with, and terminology for the user interface. Use of these rules will provide applications a consistent way to interact with users. The defaults provided by the SAA dialog services and presentation services will support these standards, making the development of applications with common user interfaces much easier. In addition, by using the SAA dialog services screen definition language, users will be able to change the defaults, thus tailoring the screens to their standards and maintaining consistency across applications.

Perhaps the most efficient way to realize the benefits of integrated applications is to use consistent formats (for example, using SAA CUA standards) for all user interactions. Terminal or printed output should be organized, using standards that make it easy for the user to identify where things are and how to interact with an application function. This standard approach to describing a user's interaction with application function will facilitate the user's transfer between applications, whether in the same or different environments, with a minimum of reorientation.

Consistent interactions require several standards. Standards should be developed for each type of display terminal. Screens should have the same layout and appearance across applications and environments. Titles, command lines, and help areas should have a standard placement.

The user's interaction with the terminal should also be consistent. Consistency here is intended to include starting a session, invoking an application, getting help or tutorial information, and making choices within applications. Commands which perform the same function should use the same names, syntax, and semantics.

Common terminology among applications will also increase productivity. In the past, this was not as important because users accessed few applications, which in many instances were designed and developed for a specific job. Today, word processing, spreadsheets, decision support, and mail services are used with great frequency, in addition to users' job-specific applications. It is perplexing and confusing to the end user when applications use the same term to mean different things and perform different functions, or when they use different terms to articulate the same meaning.

Common standards make it easier for a user to identify and invoke required actions and request assistance for new applications.

Common standards make it easier for a user to identify and invoke the required actions and to request assistance for new applications. As users gain experience with an application, functions contained in SAA dialog services will permit the bypassing of application menus designated for the novice user. The CUA publication[2] defines a consistent terminology for cross-applications and cross-systems functions, but it cannot be specific as to the terminology of each industry. Thus, application designers must also try to use standard terminology across applications.

Intelligent workstations

As indicated earlier, the presentation manager in OS/2 supports application management functions that are extensions to the SAA presentation interface. By taking advantage of the SAA communications interface to connect to an OS/2 workstation, an application written in a cooperative fashion can have its user interface running on the OS/2 workstation, with the remainder of the processing taking place on the host system. A cooperative application can take advantage of the interactive capabilities of the workstation. In fact, the user interface can look the same as the user interface of a local application. This provides consistency for the user, regardless of the system on which the processing occurs.

Intelligent workstations can also make it possible to integrate existing applications by replacing their existing user interfaces. This is accomplished through the capability of the workstation to put computing power between the keyboard and the data input of a terminal emulator. The existing application may not need to be modified. The workstation can invoke and communicate with existing applications and present a standard interface to the user or other applications. At the same time, existing applications continue to see the data as though the user were keying them in directly.

Over time, the SAA CPIs can be extended to decrease the implementation efforts required for developing, managing, and using cooperative applications such as in the insurance environment described earlier. For example, services could be provided to set up the environment for host code automatically.

As with any higher-level interface, this has the potential to decrease the flexibility of an application. SAA has started with the most general functions and can be extended over time with more specific functions as the need arises.

Concluding remarks

This paper defines *integrated applications* as diverse functions and services that appear to the end user as a single, consistent system. The functions and services often do not have an industry or technical relationship, but they are all necessary to the end user's job performance. In the past, a typical end user had access to few computer applications, and then only those that were directly job-related. With advances in computer technology (networking, in particular), reductions in computing cost, and the availability of personal computing power, an era of more extensive computer usage is emerging. Users now have access to many general-purpose applications, such as office, document-processing, spreadsheet, and graphics applications. Collective use of these applications and job-specific applications can significantly enhance productivity. The applications must be consistent and easy to use if they are to be organized in a manner that supports the natural or prescribed way of carrying out a user's job responsibilities.

This paper discusses planned and unplanned application integration using the elements of SAA to share, access, and interchange data. Key to this discussion is SAA support for writing portable applications and locating them as close as possible to the user. Also discussed is the importance of developing applications using common functions to improve consistency. SAA provides facilities to assist in executing these functions, whether local or remote, and make the functions appear local to the end user.

The methodology of Common User Access (CUA) and the SAA elements necessary to implement it are described in terms of the importance of making applications appear consistent. Illustrating the CUA discussion is the use of intelligent workstations and the SAA elements planned for the workstation. They will help facilitate a consistent user view by providing transparency to the computer network and the many different computing environments in which user applications may actually be running.

In closing, it is important to remember that SAA definitions and products are developing. SAA will grow and evolve as requirements are identified, solutions designed, and products implemented. In the future, SAA should simplify the job of developing new applications to be integrated with one another and with the user's existing base of applications.

Cited references

1. *Systems Application Architecture—An Overview*, GC26-4341-1, IBM Corporation; available through IBM branch offices.
2. *Systems Application Architecture Common User Access Panel Design and User Interaction*, SC26-4351, IBM Corporation; available through IBM branch offices.

Lawrence A. Buchwald *IBM Application Systems Division, 472 Wheelers Farms Road, Milford, Connecticut 06460.* Mr. Buchwald joined IBM in 1982, specializing in the areas of database technology and data modeling. In 1984, he joined the information center staff at the Information Systems and Technology Group headquarters, where he was responsible for managing end-user executive services. He is currently a senior planner, with responsibilities for strategies and plans in the area of application integration. Mr. Buchwald has an M.B.A. in management science from Pace University, New York City.

Richard W. Davison *IBM Application Systems Division, 472 Wheelers Farms Road, Milford, Connecticut 06460.* Mr. Davison joined IBM in 1966 as a systems engineer trainee in the Poughkeepsie branch office. His main areas of specialty were process control and communications as applied to the generation and distribution of power in the power industry. He later joined the

Information Systems Group, participating in the development of the Virtual Storage Extended/Interactive Problem Control System (VSE/IPCS) and the Interactive System Productivity Facility (ISPF) Dialog Manager program products, in the early efforts at defining the Common User Access architecture. Mr. Davison was development manager for the MVS/Prolog program offering. He is currently a senior planner of programming architecture, with responsibility for product requirements, programming architectures, and product plans in the areas of communications and distributed processing. He has a B.S. degree in engineering science from the Rensselaer Polytechnic Institute, Troy, New York.

Wayne P. Stevens *IBM Application Systems Division, 472 Wheelers Farms Road, Milford, Connecticut 06460.* Mr. Stevens joined IBM as a systems engineer in 1967, specializing in performance analysis and improved programming techniques. He lectures and publishes on structured design and data flow, and has written on the topic of software design for a series of volumes on application development to be published by Prentice-Hall, Inc., Englewood Cliffs, New Jersey, in 1988. Mr. Stevens is currently responsible for planning future IBM application development products.

Reprint Order No. G321-5328.